PARTITIONED CONVOLUTION IN NEURON NETWORK

PERFORMANCE IMPACT ON NEURAL NETWORK WITH PARTITIONED CONVOLUTION IMPLEMENTED WITH GPU PROGRAMMING

By Bill Lee, B.Eng

A Thesis Submitted to the School of Graduate Studies in Partial Fulfilment of the Requirements for the Degree Master of Applied Science in Software Engineering

MASTER OF APPLIED SCIENCE IN SOFTWARE ENGINEERING (2021)

Department of Computing and Software,
McMaster University, Hamilton, Ontario

TITLE: Performance Impact on Neural Network with Partitioned Convolution

Implemented with GPU Programming

AUTHOR: Bill Lee, B.Eng. Mechatronics (McMaster University)

SUPERVISOR: A. Wassyng and M. Lawford
NUMBER OF PAGES: xi, 65

Lay Abstract

A convolutional neural network is a machine learning tool that allows complex patterns in datasets to be identified and modelled. For datasets with input that consists of the same type of data, a convolutional neural network is often architected to identify global patterns. This research explores the viability of partitioning input data into groups and processing them with separate convolutional layers so unique patterns associated with individual subgroups of input data can be identified. The author of this research built suitable test datasets and developed a (parallel computation enabled) framework that can construct both standard and proposed convolutional neural networks. The test results show that the proposed structure is capable of performance that matches its standard counterpart. Further analysis indicates that there are potential methods to further improve the performance of partitioned convolution, making it a viable replacement or supplement to standard convolution.

Abstract

For input data of homogenous type, the standard form of convolutional neural network is normally constructed with universally applied filters to identify global patterns. However, for certain datasets, there are identifiable trends and patterns within subgroups of input data. This research proposes a convolutional neural network that deliberately partitions input data into groups to be processed with unique sets of convolutional layers, thus identifying the underlying features of individual data groups. Training and testing data are built from historical prices of stock market and preprocessed so that the generated datasets are suitable for both standard and the proposed convolutional neural network. The author of this research also developed a software framework that can construct neural networks to perform necessary testing. The calculation logic was implemented using parallel programming and executed on a Nvidia graphic processing unit, thus allowing tests to be executed without expensive hardware. Tests were executed for 134 sets of datasets to benchmark the performance between standard and the proposed convolutional neural network. Test results show that the partitioned convolution method is capable of performance that rivals its standard counterpart. Further analysis indicates that more sophisticated method of building datasets, larger sets of training data, or more training epochs can further improve the performance of the partitioned neural network. For suitable datasets, the proposed method could be a viable replacement or supplement to the standard convolutional neural network structure.

ACKNOWLEDGEMENTS

I would like to thank my supervisory team, Dr. Alan Wassyng, Professor, Department of Computing and Software and Dr. Mark Lawford, Chair, Department of Computing and Software, also Dr. Wenbo He, Associate Professor, Department of Computing and Software for their support, feedback, and insightful guidance throughout the writing of this thesis.

Much appreciation to everyone who contributed to this research, including all those who are responsible for making vast number of resources available online.

I would like to dedicate this work to my family: my wife Yvonne for her support and encouragement, as well as Joshua and Benjamin, my sons who were born during my time pursuing this degree. You have always been my source of motivation to go further in life.

Table of Contents

Table of Figures

Table of Tables

List of Abbreviations and Symbols

CNN	Convolutional Neural Network
PCNN	Partitioned Convolutional Neural Network
CPU	Central Processing Unit
GPU	Graphics Processing Unit
CUDA	Compute Unified Device Architecture
ReLU	Rectified Linear Unit
API	Application Programming Interface

DECLARATION OF ACADEMIC ACHIEVEMENT

I, Bill Lee, declare this thesis to be my own work. I am the sole author of this document. No part of this work has been published or submitted for publication or for a higher degree at another institution.

To the best of my knowledge, the content of this document does not infringe on anyone's copyright.

My supervisors, Dr. Alan Wassyng and Dr. Mark Lawford, and the member of my supervisory committee, Dr. Wenbo He, have provided guidance and support for this paper. I completed all the research work.

1 Introduction

In the field of machine learning, Convolutional Neural Network (CNN) is a fundamental concept that enables deep learning. It is a feed-forward neural network with many parameters in hidden layers that typically alternate between a convolutional layer and a subsampling (pooling) layer (Nielsen, 2015). It has a wide range of applications and is practically the standard for machine learning operations in several industries (Serkan Kiranyaz, 2019). Naturally, there are ample efforts to improve its performance and broaden its applications.

There are several variations on how convolutional layers in neural network can be setup. These variations generally depend on the nature of the data in the modelled datasets, especially the input data. This research is based on the notion that there could be more than one suitable neural network structure. For a set of datasets that can be modelled by standard CNN, the author of this document proposes that it may also be modelled by deliberately partitioning the convolution process in the neural network. This research explores the viability of this approach. It is theorized that for some datasets, the proposed method of convolution would match and, in some cases, surpass the performance of standard CNN.

This document discusses in depth the underlying concepts of the proposed approach and its potential applications. The author designed and implemented a software framework to build necessary neural networks for testing and benchmarking. Datasets suitable for both standard and proposed

convolutional networks were constructed and properly preprocessed. This document includes both methodology and result of the tests performed for this research as well as the subsequent analysis.

2 Theory and Concept

2.1 Convolutional Layer in Neural Network

Figure 1 shows a typical 2D convolutional layer in a neural network used for image classification (Yakura, March 2018). Output is the dot product between the filter values and the corresponding local values in the input map. In practice, multiple filters are often used to generate multiple channels (Nielsen, 2015). A convolutional layer has several advantages over a simple dense layer, shown in Figure 2 (Isaksson, 2020). Convolutional layers are inherently more resistant to overfitting, which means less chance of an exploding or vanishing gradient during backpropagation. More importantly, it can identify features formed by input data points while the dense layer is unable to account for the "location" of values in the input map (Nielsen, 2015).

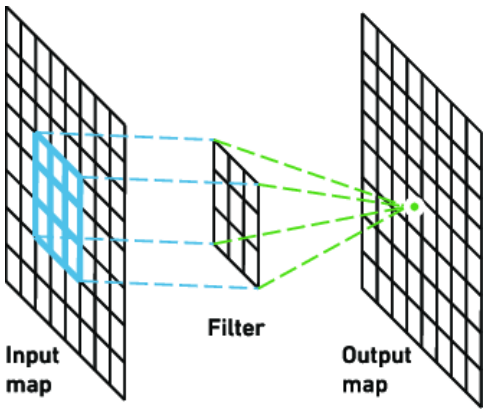

Figure 1: Convolutional Layer

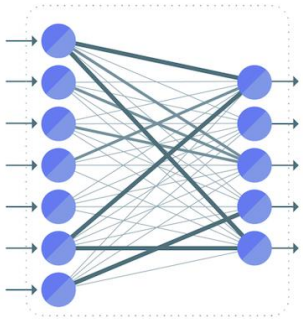

Figure 2: Dense Layer

2.2 Convolutional Strategy

Convolutional layers need to be appropriately setup according to the characteristics of the modelled data, especially input. For example, for a convolutional layer with images as its input as shown in Figure 1, there are no structural limitations to the shape of the filter since all input data are completely homogeneous with each value representing a pixel. The convolutional filter is likely a square matrix so features in the images can be adequately captured.

The convolutional layer setup would be different if the input data is comprised of heterogenous groups of data with no correlations to each other. Figure 3 represents such an example. The input data is comprised of readings over time from different types of sensors, which includes electrical current, electrical voltage, water pressure, and radiation level. In this case, each vector of sensor readings would need to be processed in separate 1D convolutional layers.

i_0	i_1	i_2	i_3	i_4	i_5
v_0	v_1	v_2	v_3	v_4	v_5
p_0	p_1	p_2	p_3	p_4	p_5
r_0	r_1	r_2	r_3	r_4	r_5

Figure 3: Heterogenous Groups of Data

Figure 4 shows a particularly interesting example of input data for a convolutional layer. There are multiple data series of the same type. There are no inter-row patterns to be identified since any two rows can switch and the overall represented data would remain the same. Therefore, the filter for convolution must be a vector (size $1 \times n$). Because all rows represent data of the same type, the same filter can be used for convolutional process of every row. In this case, global patterns and trends would be "learned" by the neural network.

$i_{A,0}$	$i_{A,1}$	$i_{A,2}$	$i_{A,3}$	$i_{A,4}$	$i_{A,5}$
$i_{B,0}$	$i_{B,1}$	$i_{B,2}$	$i_{B,3}$	$i_{B,4}$	$i_{B,5}$
$i_{C,0}$	$i_{C,1}$	$i_{C,2}$	$i_{C,3}$	$i_{C,4}$	$i_{C,5}$
$i_{D,0}$	$i_{D,1}$	$i_{D,2}$	$i_{D,3}$	$i_{D,4}$	$i_{D,5}$

Figure 4: Independent Data Groups of the Same Type

However, the approach proposed by this research is to separate this input data by row and process them with independent 1D convolutional layers in parallel. At the cost of increasing trainable parameters and not learning global patterns, this setup would allow patterns that are associated with individual groups of input data to be identified by the neural network. The research presented in this document focuses on datasets where they can be modelled by both standard convolutional layer with a universal filter or partitioned convolutional layers with individualized filters.

2.3 Prior Work

Processing a row (or vector) of data with 1D convolutional layer is not a new concept. An in-depth research survey shows that the application of 1D CNN

has seen successes in recent years (Serkan Kiranyaz, 2019). Examples of its applications include detection of adverse condition such as real-time motor fault and structural damage (Osama Abdeljaber, 2017) (Turker Ince, 2016). There has also been research on its application with sound event detection (Hyungui Lim, 2017). A 1D CNN model sometimes includes a layer to fold 2D input into 1D data, which is often done when the model is used for natural language processing (Kim, 2014).

Research related to CNN generally focuses on building a neural network structure that is best suited for specific datasets. There has not been much work on deliberately partitioning input data for separate convolutional processes, especially when the input data can be adequately modelled by traditional CNN. The convolutional structure proposed in this research also runs contrary to the general practice of minimizing trainable parameters. But some existing work is relevant to the idea of partitioning convolutional processes in a neural network. For example, one publication discussed the use of a neural network structure where results from several parallel-running CNNs are merged for downstream processing (Siavash Sakhavi, 2015). This suggests that for suitable datasets, merging the results of partitioned convolution processes, a critical characteristic of the method proposed in this research, is theoretically viable.

2.4 Proposed Model

Figure 5 and Figure 6 show the difference between a standard convolutional layer and the partitioned convolutional layer. Instead of using a

universal filter (vector of weights), a unique filter is used for each data row. It is also possible to create multiple channels of convolution by having multiple filters for one data row. Other aspects such as pooling layer, activation function and use of biases parameters do not change with the proposed partitioned convolutional layers. Figure 7 shows an example of a pooling layer.

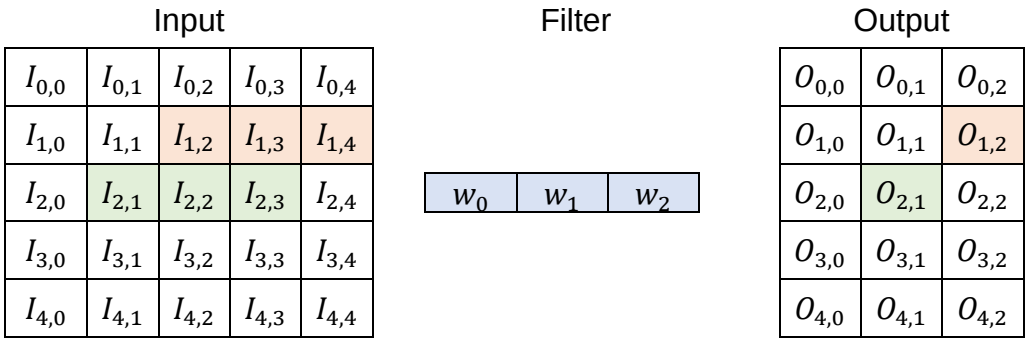

Figure 5: Standard Convolutional Layer

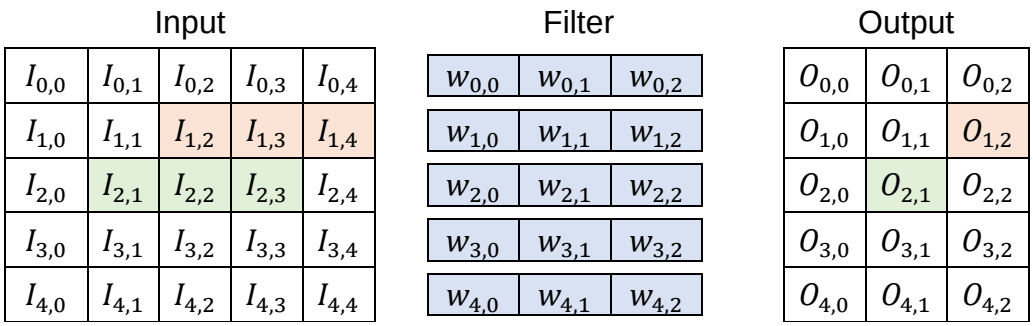

Figure 6: Partitioned Convolutional Layer

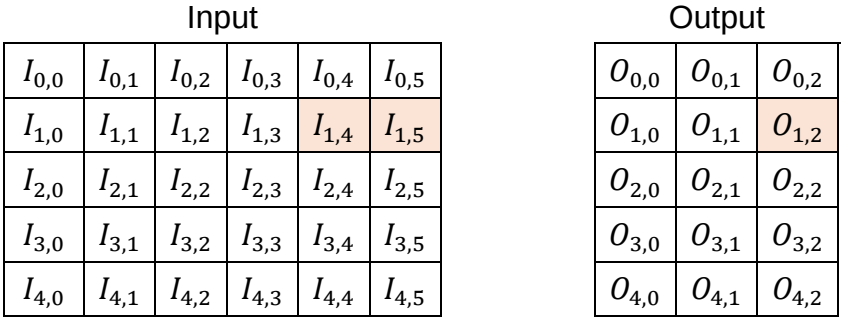

Figure 7: Partitioned Max Pooling

Figure 8 provides a comparison in structure between a standard CNN and the proposed structure. The proposed neural network separates input data into multiple data vectors. Each data group would be processed by a series of convolutional layers and pooling layers independently. The outputs would then be merged and processed by dense (fully-connected) layers to produce the final output. This structure is referred to as Partitioned Convolutional Neural Network (PCNN) in this document.

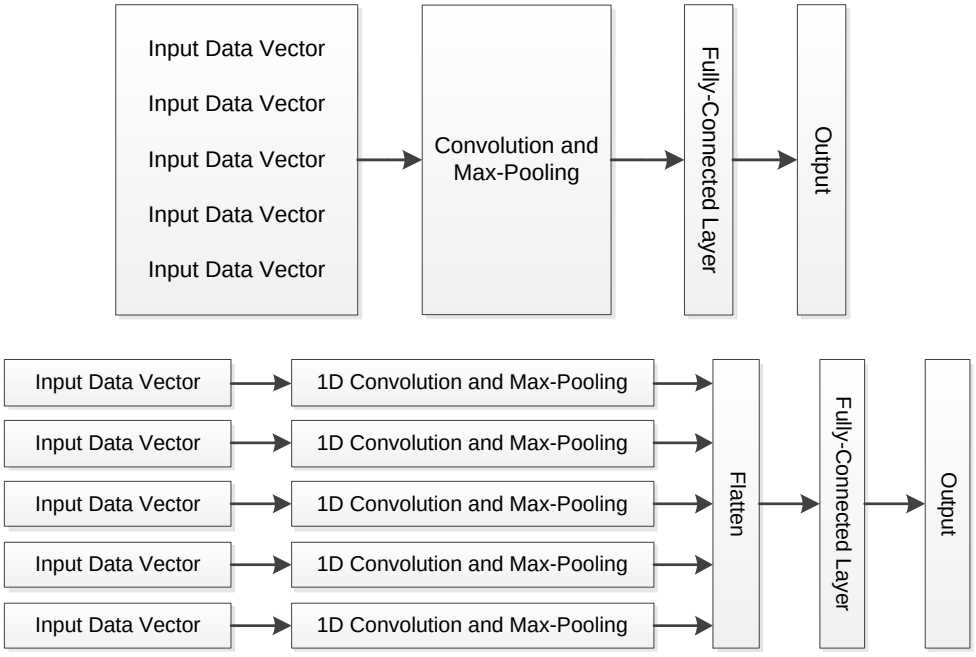

Figure 8: Structure of CNN (above) and PCNN (below)

3 Implementation

3.1 Existing Framework

Several frameworks, with complete documentation and API (Application Programming Interface), are freely available to implement machine learning models. The following frameworks were investigated:

- TensorFlow (Google Brain Team, 2020)
- Keras (Massachusetts Institute of Technology, 2020)
- PyTorch (Torch Contributors, 2020)

The frameworks listed above are all capable of constructing commonly used structures of neural networks. While modeling the partitioned convolutional layers would not be as straightforward, all investigated frameworks support modeling parallel convolutional layers. However, the unique structures of the proposed neural network may prevent the framework from optimizing for the best performance.

In addition, there are advantages in having control and transparency on the details of the implementation, especially when this research is focused on the low-level logic of neural network. Thus, the author of this research decided to develop a specific API to construct neural networks and to run any necessary tests.

3.2 Calculation

Information on calculation related to neural network is readily available (Nielsen, 2015). PCNN proposed in this document shares many identical operations and processes with standard CNN. Upon input data partitioning at the beginning of the neural network, each partitioned group is then processed by unique series of layers of the same configuration. The scope of this research focuses on partitioned 2D input data into 1D vector groups.

Figure 9 illustrates input, filter, and output of a convolutional layer to process a data vector. In this example, there are three channels (or depth) for input data and two channels for output data. This corresponds to $3 \times 2 = 6$ filter vectors.

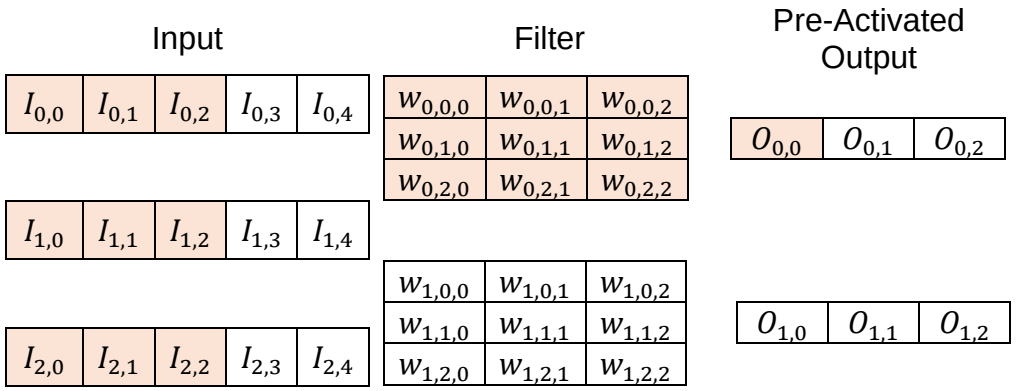

Figure 9: Forward Calculation for a Single Data Row

The calculation for output $O_{0,0}$ (as highlighted in Figure 9) is:

$$I_{0,0} \times w_{0,0} + I_{0,1} \times w_{0,1} + I_{0,2} \times w_{0,2} + I_{1,0} \times w_{1,0} + I_{1,1} \times w_{1,1} + I_{1,2} \times w_{1,2} + I_{2,0} \times w_{2,0} + I_{2,1} \times w_{2,1} + I_{2,2} \times w_{2,2}$$

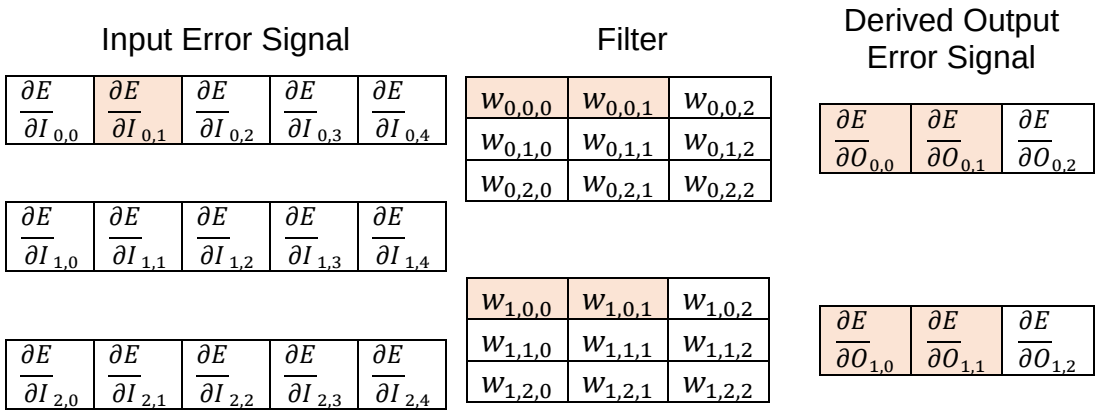

Figure 10: Backpropagation for a Single Data Row

For backpropagation, as shown in Figure 10, the calculation for $\frac{\partial E}{\partial I_{0,1}}$ is:

$$\frac{\partial E}{\partial O_{0,0}} \times w_{0,0,1} + \frac{\partial E}{\partial O_{0,1}} \times w_{0,0,0} + \frac{\partial E}{\partial O_{1,0}} \times w_{1,0,1} + \frac{\partial E}{\partial O_{1,1}} \times w_{1,0,0}$$

The calculation for gradient of $w_{0,2,0}$ is:

$$\frac{\partial E}{\partial I_{0,0}} \times \frac{\partial E}{\partial O_{1,0}} + \frac{\partial E}{\partial I_{0,1}} \times \frac{\partial E}{\partial O_{1,1}} + \frac{\partial E}{\partial I_{0,2}} \times \frac{\partial E}{\partial O_{1,2}}$$

3.3 Features

There are several factors to consider when implementing the API. A neural network can model output data of both regression and classification. To keep the implementation simple, the API focuses on modelling classification datasets. A regression dataset can be converted into a classification dataset (Brownlee, 2017). This means the error function for backpropagation would be Cross Entropy (Nielsen, 2015). The Adam optimization algorithm is selected for its fast convergence and built-in regulation features (Brownlee, 2017). Training is done with mini-batch gradient descent to obtain balance between convergent speed, computation efficiency and learning noise minimization (Brownlee, 2017).

Some features were considered but not included. Dropout can create a "voting" mechanism within the network while providing regulation during training (Nielsen, 2015). Batch normalization greatly reduces the possibility of vanishing or exploding gradient while preventing overfitting (Brownlee, 2019). However, both features would add more trainable parameters to the network and place higher requirements on the number of training datasets. Therefore, these two features were not implemented for the API. The research thus relied on preprocessing of input data to achieve similar benefits.

Both standard CNN and proposed PCNN are to be built with the developed API. This enables benchmarking between two methods of convolutions with the same test datasets.

3.4 Parallel Programming

Machine learning and especially CNN is computationally heavy. Time to execute tests would be significant if the implementation handles all calculations with a Central Processing Unit (CPU). Instead of looking for expensive hardware, parallel programming is utilized to take advantage of the parallelable nature of the calculations.

A laptop, Lenovo Legion Y520, was available for the research. The relevant hardware specifications are listed in Table 1. Parallel programming with a Graphics Processing Unit (GPU) is enabled by Compute Unified Device Architecture (CUDA) framework. Documents for CUDA are readily available from Nvidia (NVIDIA Corporation, 2017).

Component	**Specification**
Processor	7th Generation Intel® Core™ i7-7700HQ Processor (2.80GHz, up to 3.80GHz with Turbo Boost, 6MB Cache)
Operating System	Windows 10 Home
Graphics	NVIDIA® GeForce® GTX 1050Ti 4 GB
Memory	16 GB DDR4 2400 MHz

Table 1: Relevant Specifications of Lenovo Legion Y520

3.4.1 Optimization for Hardware

CUDA Occupancy Calculator is available from NVIDIA to help with the architecture design of the implementation (NVIDIA Corporation, 2017). Though

the computation load for neural network is heavy, the calculation logic itself is simple and does not use a high number of registers. Thus, the thread block sizes of 256, 512 or 1,024 are all acceptable according to the analysis tool. A thread block size of 512 is chosen for the implementation.

Another design decision is into how many threads should a calculation task be partitioned. It has been suggested that there should be at least 14,000 threads running concurrently for Tesla K20X, which has 2,688 CUDA cores (Woolley, 2013) (NVIDIA Corporation, 2013). GTX 1050Ti GPU has 768 CUDA cores (NVIDIA Corporation, 2019). It is determined that the implementation would attempt to achieve at least **8,192** total threads. It is important to note that separating calculation into too many threads could hurt the performance of certain operations, as shown in later sections of this document.

3.4.2 Calculation

Table 2 shows the required calculations for both forward computation and backpropagation in a convolutional neural network. Some operations are simple to parallelize, such as:

- Output error signal calculation
- Parameter updates with gradients

However, most calculations involve a reduction (by addition) operation. A simple approach is to have one thread handling all calculations for an element in the output vector or matrix. This approach has the advantage of simplicity in its implementation. If the output vector or matrix is large, the calculation would be

naturally partitioned into enough number of threads to achieve good parallel computation performance. However, some operations would not meet this criterion, such as the weight gradient calculation for a convolutional layer or calculations related to dense layers. A more dynamic method to setup threads is needed to better utilize the parallel computation potential of the GPU.

Operation		**Calculation Required**
Forward Calculation	Convolutional Layer	- Matrix multiplications - Reduction (by addition) - Apply bias and activation
	Pooling Layer	- Reduction
	Dense (Fully Connected) Layer	- Matrix multiplications - Reduction (by addition) - Apply bias and activation
Backpropagation	Output Error Signal Calculation	- N-to-N vector mapping
	Dense (Fully Connected) Layer	- Derive error signals - Matrix multiplications - Reduction (by addition)
	Pooling Layer	- Un-reduction
	Convolutional Layer	- Derive error signals - Matrix multiplications - Reduction (by addition)
	Weight Gradient Calculation	- Matrix multiplications - Reduction (by addition)
	Parameter Updates with Gradient	- Matrix addition

Table 2: Calculation Required in PCNN

3.4.3 Partition reduction operation

To reach the desired total thread count of 8,192, the process to calculate a single output value often need to be partitioned into multiple threads. The

implementation is heavily inspired by a guide on optimizing parallel reduction in CUDA (Harris, 2007). The overall principles for the implementation include:

- Avoid writing to or reading from GPU memory except for data that needs to persist outside of the operation
- Use shared memory for inter-thread communication required by reduction operation to minimize latency (Harris, 2013)
- Perform part of reduction operation at thread-level since a register is faster than shared memory

Figure 11 shows the thread-level logic flow when a reduction operation is separated into 4 threads. Assuming the logic requires preliminary calculation to obtain the 23 inputs for the reduction operation and post-reduction calculation to obtain the final output:

- Each thread would perform pre-reduction calculation for 6 (or 5) of the inputs of the reduction operation
- Each thread would perform thread-level reduction of the 6 (or 5) inputs it has calculated
- The threads are synchronized for further reduction operation where shared memory is used to store temporary variable
- For 4 threads, $\log_2 4 = 2$ synchronized calculations complete the reduction operation
- The first (index 0) thread applies any post-reduction calculation on the output before storing it to GPU memory

It is worth noting that calculations should only be partitioned so the overall thread count could reach the desired 8,192. As seen in Figure 11, there is significant thread idleness after pre-reduction calculation. Shared memory also has higher latency than registers. Therefore, unnecessarily partitioning calculation into many threads would decrease performance.

	Thread #0	**Thread #1**	**Thread #2**	**Thread #3**
Thread-Level Processing	Calculate then reduce (obtain total) elements of index $8n$, stored to shared memory index 0	Calculate then reduce (obtain total) elements of index $8n + 1$, stored to shared memory index 1	Calculate then reduce (obtain total) elements of index $8n + 2$, stored to shared memory index 2	Calculate then reduce (obtain total) elements of index $8n + 3$, stored to shared memory index 3
Inter-thread Reduction	Shared memory $el[0] \mathrel{+}= el[2]$	Shared memory $el[1] \mathrel{+}= el[3]$		
	Shared memory $el[0] \mathrel{+}= el[1]$			
Post-Reduction Processing	Apply needed calculation to result at shared memory $el[0]$ then store to output vector			

Figure 11: CUDA Reduction With 8 Threads

3.4.4 Performance

The performance of the GPU-based parallel-programming implementation was benchmarked against a CPU-based implementation. The neural network structure described in Section 6.1 was built with the developed GPU-based API. A set of the datasets described in Section 5 was used as a sample. The execution time of one training epoch (with approximately 6,000 training sets) for the sample datasets was 0.04590681 second. A simple program was written to

simulate the same amount of floating-point calculation performed by a CPU, which resulted in an execution time of 5.7944 second. The simulation does not consider the higher memory latency when using a CPU. Therefore, even conservatively, the CUDA implementation has a performance advantage by a factor of 126 over a single-CPU implementation.

3.5 Implementation of Neural Network API

Nvidia-provided APIs for CUDA are all module based instead of being class based. The API was developed to implement CUDA-based calculations specific to neural network calculation through C++ classes, thus enabling object-oriented design at top level (Network class) of the API. The API allows users to build a neural network with standard convolutional layer, proposed partitioned convolutional layer, pooling layer and dense (fully-connected) layer. Figure 12 shows the dependency graph between various components that builds the Network class at the top level. The entirety of source code of the API is available at https://github.com/BillLee-SDE/Parallel/tree/master/network_API.

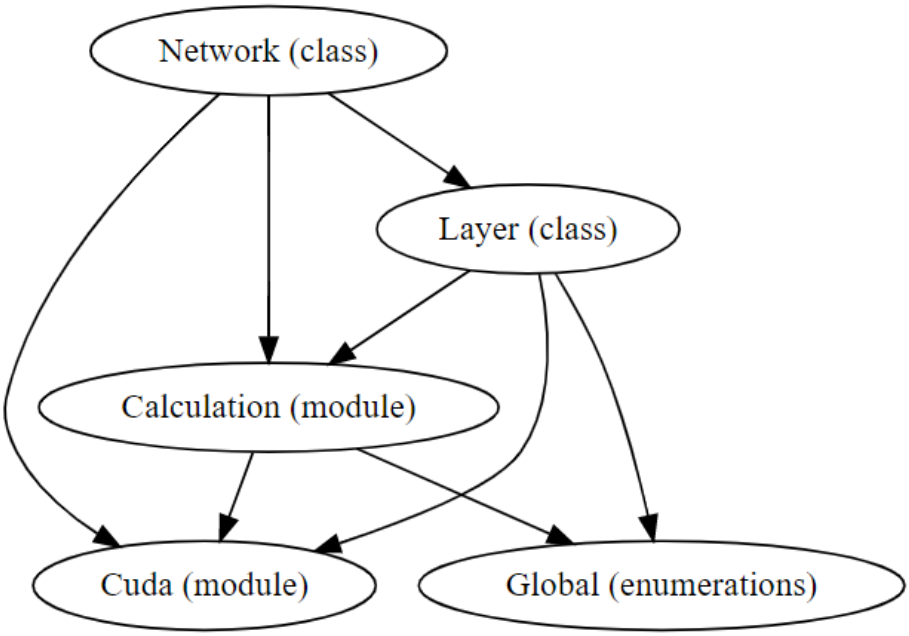

Figure 12: Internal Structure of API

3.5.1 Global (Enumeration)

This is a collection of enumerations used across the API. These enumerations are also visible to the user of the API. The code snippet below is one of the enumeration definitions.

```
enum ActivationFunction
{
        TanH = 0,
        Sigmoid = 1,
        ReLu = 2,
        LeakyReLu = 3
};`
```

3.5.2 Cuda (module)

This module is implemented with CUDA file types (cuda.cu and cuda.cuh). The module includes functions that allocate, free, and reset GPU memory. The code snippet below is the function to allocate memory.

```
template <typename T>
T* CUDA_Array_Allocate(int size)
{
        size_t free_t, total_t;
        cudaMemGetInfo(&free_t, &total_t);

        T* result;
        cudaError cudaStatus = cudaMalloc((void**)&result, size * sizeof(T));
        if (cudaStatus != cudaSuccess)
        {
                int i = 0;
                i++;
        }
        return result;
}
```

There are also several utility functions to copy data between GPU memory and regular memory, initialize GPU memory or obtain debugging information. This module also defines a class that regulates the parameters of CUDA kernel call, so a suitable number of threads is used to maximize GPU computing utilization, as described in Section 3.2. Below is the declaration of the class.

```
class CudaParameter
{
public:
        int CountBlock;
        int ThreadPerResult;

        CudaParameter(int countResult, int countCalculationPer);
};
```

3.5.3 Calculation (module)

This module is implemented with CUDA file types (calculation.cu and calculation.cuh) and include all the CUDA logic related to calculations for neural network. This includes functions that handle both forward and backpropagation of dense layer, partitioned convolutional layer, standard convolutional layer, and pooling layer. Most of the complexity in the API is implemented in this module. The code snippet below is the function that handles inter-threaded reduction shown in Figure 11. It is executed and invoked at GPU level.

```
__device__ void cuda_Reduce(
        double* buffer, double input, int threadID, int count, double* result)
{
        int id = threadID % count;

        if (count <= 1)
        {
                *result = input;
                return;
        }

        buffer[id] = input;

        if (count > 128)
        {
                __syncthreads();
                if (id >= 128 && id < 256) { buffer[id - 128] += buffer[id]; }
        }

        if (count > 64)
        {
                __syncthreads();
                if (id >= 64 && id < 128) { buffer[id - 64] += buffer[id]; }
        }

        if (count > 32)
        {
                __syncthreads();
                if (id >= 32 && id < 64) { buffer[id - 32] += buffer[id]; }
        }

        if (count > 16)
        {
                __syncthreads();
                if (id >= 16 && id < 32) { buffer[id - 16] += buffer[id]; }
        }

        if (count > 8)
        {
                __syncthreads();
                if (id >= 8 && id < 16) { buffer[id - 8] += buffer[id]; }
        }

        if (count > 4)
        {
                __syncthreads();
                if (id >= 4 && id < 8) { buffer[id - 4] += buffer[id]; }
        }

        if (count > 2)
        {
                __syncthreads();
                if (id >= 2 && id < 4) { buffer[id - 2] += buffer[id]; }
        }

        if (count > 1)
        {
                __syncthreads();
                if (id >= 1 && id < 2) { buffer[id - 1] += buffer[id]; }
        }

        __syncthreads();
        if (id == 0) { *result = buffer[0]; }
}
```

Below is the kernel function that handles forward calculate of a fully connected layer. The function is invoked at CPU level and executed at GPU level.

```
__global__ void cuda_Calculation_FullyConnected_Forward(
        int sizeBatch, int sizeInput, int sizeOutput, int* dropoutSelect,
        double* input, double* weight, double scaleRatio, double* bias,
        Global::ActivationFunction af, double* activated, int threadPerResult)
{
        extern __shared__ double buffer[];
        int index = blockDim.x * blockIdx.x + threadIdx.x;

        int indexOutputBatch = 0;
        int indexCalculation = 0;
        if (!cuda_Reduce_Setup(index, sizeBatch * sizeOutput, threadPerResult,
            &indexOutputBatch, &indexCalculation))
        {
                cuda_Reduce_Dummy(threadPerResult);
                return;
        }

        int indexBatch = indexOutputBatch / sizeOutput;
        int indexOutput = indexOutputBatch % sizeOutput;
        double result = 0.0;

        for (int i = indexCalculation; i < sizeInput; i += threadPerResult)
        {
                if (dropoutSelect == nullptr ? true : dropoutSelect[i])
                {
                        result += input[indexBatch * sizeInput + i] *
                                  weight[indexOutput * sizeInput + i];
                }
        }

        cuda_Reduce(buffer, result, threadIdx.x, threadPerResult, &result);

        if (indexCalculation == 0)
        {
                if (bias == nullptr)
                {
                        activated[indexBatch * sizeOutput + indexOutput] =
                                cuda_activation(result * scaleRatio, af);
                }
                else
                {
                        activated[indexBatch * sizeOutput + indexOutput] =
                                cuda_activation(result * scaleRatio +
                                bias[indexOutput], af);
                }
        }
}
```

The code snippet below shows how the above function is invoked.

```
cuda_Calculation_FullyConnected_Forward
        <<<parameter.CountBlock, CUDA_CALCULATION_BLOCK_THREAD_SIZE,
           CUDA_CALCULATION_BLOCK_THREAD_SIZE * sizeof(double)>>>
        (sizeBatch, sizeInput, sizeOutput, dropoutSelectInput, input, weight[0], scale, bias[0],
         af, activated, parameter.ThreadPerResult);
```

3.5.4 Layer (class)

Layer is an abstract class that defines the overall behavior of a layer in the neural network. The following child classes are defined from the abstract class:

- Input (represents the first layer that contains the input data)
- CalculationC_1D (partitioned convolutional layer)
- Pooling_1D (partitioned pooling layer)
- CalculationC_2D (standard convolutional layer)
- Pooling_2D (standard pooling layer)
- CalculationFC (fully connected layer)

The implementation of each class is based on the expected behavior of the corresponding layer. Member functions Forward() and Backward() of each class call the corresponding functions defined in Calculation module.

3.5.5 Network (class)

This is the top-level class that integrates all classes and modules together. The public functions of this class are visible as the API features, including:

- Network (constructor function that build the neural network layers based on parameters)
- AddTrainingSet (add a pair of input and output as training data)
- Error (provide pairs of input and output to calculate error of the network)
- Calculate (provide a list of input data for outputs calculated by the network)

4 Preliminary Test

Before running large-scaled experiments with real data, it is beneficial to test with an established set of datasets specifically tailored for machine learning. This serves as a sanity check to ensure the network is implemented correctly.

The MNIST database of handwritten digits was used for this test (Yann LeCun, 2020). A dataset contains 28×28 images of handwritten digits and corresponding correct categorizations. There are 50,000 training datasets and 10,000 testing datasets.

4.1 Setup

With the implemented API (described in Section 3.5), neural networks of different types can be created. The input data entries are easily standardized to values between 0 and 1 (division by 255). The initialization of weight and bias are done with the following formulas (Nielsen, 2015):

$$weight = \frac{normal\_distribution\_randomization(0,1)}{\sqrt{size\ of\ input}}$$

$$bias = 0$$

the numerator for individual weight initialization is the result of normal distribution randomization which is centered at 0 with standard deviation of 1. The denominator is the square root of the size of input vector or matrix. All biases are simply set to 0 at the beginning.

Three separate neural networks were built to benchmark the performance of dense-only, CNN and PCNN with the datasets.

Dense-Only Neural Network		
Layer	**Parameters**	**Output Size**
--	--	$28 \times 28 = 784$
Dense	$784 \times 512 = 401{,}408$	512
Dense	$512 \times 10 = 5{,}120$	10

Convolutional Neural Network		
Layer	**Parameters**	**Output Size**
--	--	$28 \times 28 = 784$
Convolutional Layer	20 channels of 5×5 filters $5 \times 5 \times 20 = 500$ parameters	$24 \times 24 \times 20 = 11{,}520$
Max Pooling Layer	Pooling by 2×2 No parameters	$12 \times 12 \times 20 = 2{,}880$
Convolutional Layer	40 channels of 5×5 filters $5 \times 5 \times 40 = 1{,}000$ parameters	$8 \times 8 \times 40 = 2{,}560$
Max Pooling Layer	Pooling by 2×2 No parameters	$4 \times 4 \times 40 = 640$
Dense	$640 \times 512 = 327{,}680$	512
Dense	$512 \times 10 = 5{,}120$	10

Partitioned Convolutional Neural Network		
Layer	**Parameters**	**Output Size**
--	--	$28 \times 28 = 784$
Partitioned Convolutional Layer	20 channels of 5 filters with size of 1×28 $5 \times 28 \times 20 = 2{,}800$ parameters	$24 \times 28 \times 20 = 13{,}440$
Max Pooling Layer	Pooling by 2×1 No parameters	$12 \times 28 \times 20 = 6{,}720$
Partitioned Convolutional Layer	40 channels of 5 filters with size of 1×28 $5 \times 28 \times 40 = 5{,}600$ parameters	$8 \times 28 \times 40 = 8{,}960$
Max Pooling Layer	Pooling by 2×1 No parameters	$4 \times 28 \times 40 = 4{,}480$
Dense	$4{,}480 \times 512 = 2{,}293{,}760$	512
Dense	$512 \times 10 = 5{,}120$	10

4.2 Result

All three networks were trained with 50,000 training datasets, mini-batch size of 64 and 20 epochs. Activation function is Rectified Linear Unit (ReLU) for

all layers except the final dense layer where Sigmoid is used. The trained networks were then used to categorize 10,000 test sets, which were compared against expected outputs.

Network Type	**Only Dense**	**CNN**	**PCNN**
Run #1	9824	9880	9785
Run #2	9824	9880	9778
Run #3	9837	9906	9814
Run #4	9831	9908	9791
Run #5	9813	9904	9762
Average	***9826***	***9896***	***9786***

Table 3: Preliminary Test Result of 10,000 Datasets

Table 3 shows the result of the preliminary test. The follow observations and analysis are made:

- Standard CNN yielded the best performance in accuracy, averaging 9896 out of 10000. It is also the only network capable of exceeding 99% accuracy in a test run. This demonstrates the advantage of identifying features formed by input data points.
- PCNN had the lowest accuracy. Multiple reasons likely contributed to this result.

Figure 13 illustrates the difference of filter usage between CNN and PCNN. For 2D image categorization, all input data points are homogenous. Therefore, 2D filter should be used with standard CNN to identify local features in particular areas of the image. On the other hand, deliberately partitioning the input image into rows and using vector filters make little sense since the network would be unable to recognize any features formed by pixels from different rows.

Therefore, the partitioned convolutional layers add no real value except creating more noise. Furthermore, adding two partitioned convolutional layers introduces additional 1,900,752 parameters, thus making the network harder to train. This is clearly a scenario where PCNN is not suitable.

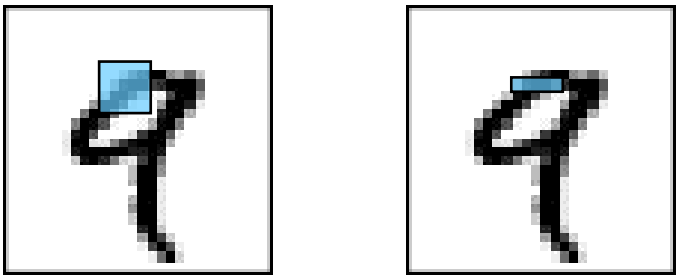

Figure 13: CNN (left) vs PCNN (right) for image recognition

Three sample networks in the preliminary test produced results that are expected. This indicates the API implementation described in Section 3.5 was working as intended and ready for testing with real data.

5 Test Data

There are a few criteria for test data selection. The most important one is that the input of the dataset must be suitable for both standard CNN and proposed PCNN. The number of datasets must also be large enough to facilitate training.

5.1 Existing sources

There are numerous datasets available for the purpose of machine learning testing and benchmarking. However, most of these data libraries are specifically prepared to test standard machine learning network models and may not be suitable for the proposed PCNN.

5.2 Data Selection

Stock movement analysis is a common application of machine learning. It is also suitable in this research for the following reasons:

- Data is readily available.
- The size of data is large enough as a lot of stocks have price history that go back more than 10 years.
- Prices of different stocks in the same market share an underlying correlation, which enables them to be modelled by standard CNN. But they also have no spatial relationship with each other when represented as rows in a matrix. For example, Figure 14 and Figure 15 are the same. This means that they are suitable for PCNN modelling.

$p_{0,0}$	$p_{0,1}$	$p_{0,2}$	$p_{0,3}$	$p_{0,4}$	$p_{0,5}$
$p_{1,0}$	$p_{1,1}$	$p_{1,2}$	$p_{1,3}$	$p_{1,4}$	$p_{1,5}$

Figure 14: Input #1 for PCNN

$p_{1,0}$	$p_{1,1}$	$p_{1,2}$	$p_{1,3}$	$p_{1,4}$	$p_{1,5}$
$p_{0,0}$	$p_{0,1}$	$p_{0,2}$	$p_{0,3}$	$p_{0,4}$	$p_{0,5}$

Figure 15: Input #2 for PCNN

It was assumed that, on a given date (the pivot date), price movement of multiple stocks in the past 120 days can be used to model the movement of a single stock in the next 30 days. For this research, 40 stocks (including the forecasted stock) are used to create input data.

5.3 Data Acquisition

As mentioned in the previous section, there are several online sources to obtain stock price history. The North America stock market was selected as the source. As the first step, a database was built through the following steps:

- The stock list was obtained from www.eoddata.com for (EODData, LLC., 2019):
    - New York Stock Exchange
    - American Stock Exchange
    - NASDAQ Stock Exchange
    - Toronto Stock Exchange
- With use of a scripting language, historical prices for each stock from the lists obtained in the previous step are downloaded from Yahoo or Alpha Vintage (Yahoo!, 2019) (Alpha Vantage Inc., 2019). The downloaded data is stored in a SQL database.

5.4 Dataset Construction and Pre-processing

5.4.1 Predictor Stock Selection

One needs to consider how to select 39 stocks in addition to the forecasted stock to build the input matrix. There must be enough historical data of every selected stock to build enough datasets. It was decided that the historical data of a stock must be up to date and go back more than 3,653 days. Another consideration is that if the prices of two stocks are highly correlated, they will be supplying redundant information as input. Therefore, the selection process is as follow:

1. Start with the stock to be forecasted.
2. Randomly pick another stock.

3. If the newly picked stock does not have enough data, discard, and go to Step 2.
4. If the newly picked stock has a correlation coefficient that is greater than 0.7 with any stocks that are already selected, discard and go to Step 2.
5. Keep the selected stock and return to Step 2 until 40 stocks have been selected.

To speed up the individual query, a SQL table was created to store pre-calculated correlation coefficients of all possible stock pairs. The logic of selecting 39 suitable predictor stocks (based on the steps described above) is implemented in a SQL store procedure.

5.4.2 Representing Price Movement

It is simple to build a dataset input with raw stock price. But there are several problems with this approach. The price of a stock can be a few cents or thousands of dollars. This would make it difficult to initialize training parameters, specifically the weights. If initial weights are too low, the training speed of the network would take too long to ramp up. On the other hand, if weights are too high at initialization, exploding and vanishing gradients would occur, creating large number of "dead" nodes and effectively crippling the network.

The solution is simple. Instead building datasets with raw stock price, the daily change in price, represented by a percentage, is used. Therefore, each data point is calculated from the formula below:

$$\frac{price_{current\ day}}{price_{previous\ day}} - 1$$

Figure 16 shows the format of the output when the SQL stored procedure mentioned in the previous section is called. In the first data column, $c_{0,n}$ represents the daily price change of the forecasted stock, which also acts as a predictor.

$$\begin{bmatrix} date_0 & c_{0,0} & c_{1,0} & \cdots & c_{39,0} \\ date_1 & c_{0,1} & c_{1,1} & \cdots & c_{39,1} \\ date_2 & c_{0,2} & c_{1,2} & \cdots & c_{39,2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ date_n & c_{0,n} & c_{1,n} & \cdots & c_{39,n} \end{bmatrix}$$

$c_{m,n}$ represents a daily price change calculated from $\frac{price_{\ stock_m, date_n}}{price_{stock_m, date_{n-1}}} - 1$

Figure 16: Data structure of one dataset

5.4.3 Building Datasets

Figure 17 provides a visual representation on how a dataset is built from the data represented by Figure 16. The input matrix is obtained by taking a 40×120 block from the data matrix, represented by the blue section in Figure 17. The output is constructed by reducing the 1×30 vector from the first column (represented by the yellow section in Figure 17) immediately after the input matrix. The formula to reduce the 1×30 vector is:

$$Output = \left(\sum (c + 1)\right) - 1$$

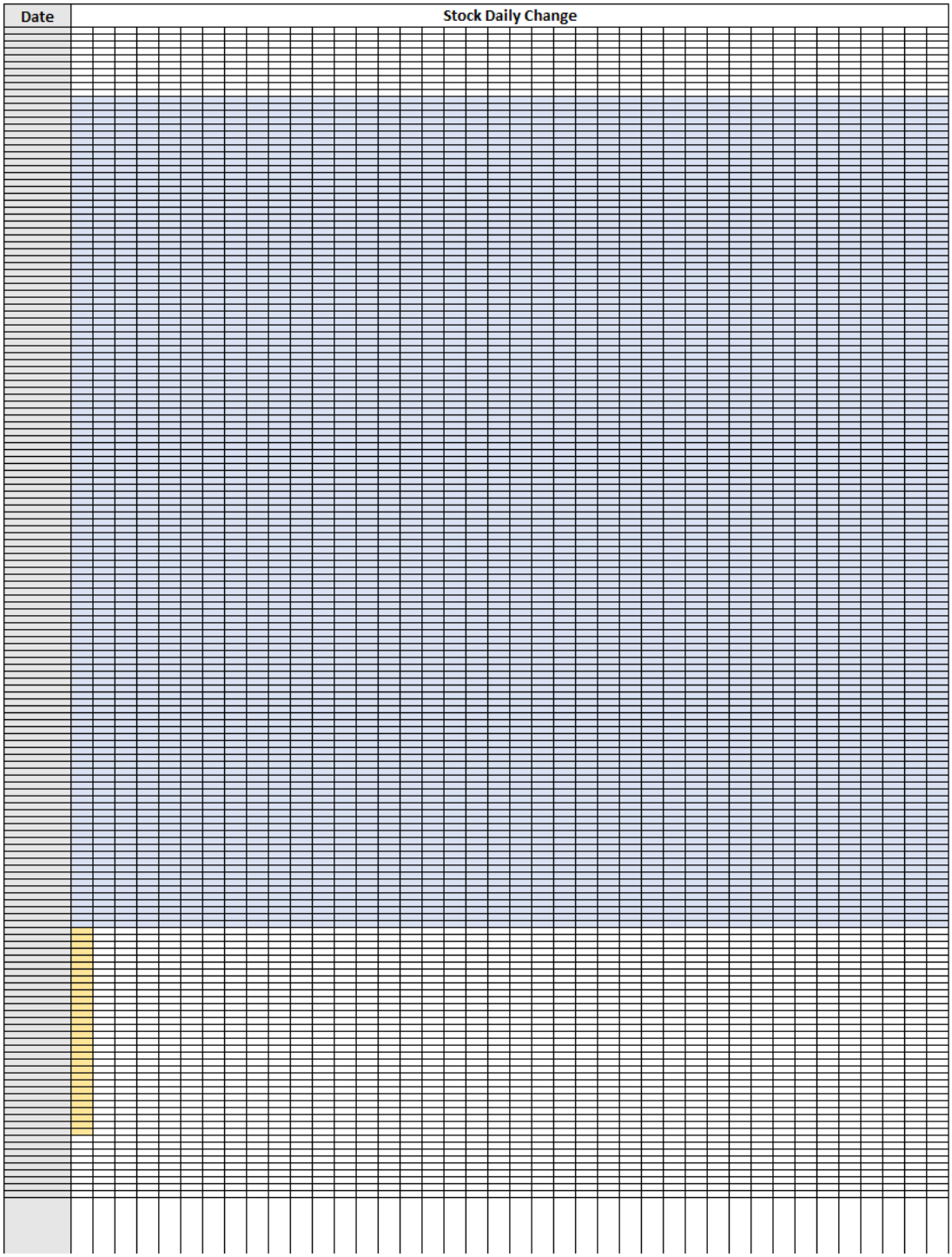

Figure 17: Building a dataset from SQL output matrix

Intuitively, the date corresponding to the last row of the input matrix can be considered the "pivot date". The input matrix represents the daily price change of

40 stocks (1 forecasted stock and 39 predictor stocks) during the 120-days period up to the pivot date. The output is a percentage representing the overall movement of the forecasted stock 30 days after the pivot day. A unique dataset is generated for each pivot day. If there are 7,000 days (rows) in the data matrix shown in Figure 16, then $7{,}000 - 120 - 30 + 1 = 6{,}851$ datasets will be generated.

As mentioned in Section 3.3, classification is preferred over regression to enable the use of ReLU activation function and max pooling. Therefore, output is further transformed from a percentage to a vector by applying the logic defined in Table 4. Standard deviation (σ) is calculated from the percentage-based output of all generated datasets.

Range of Stock Change	**Stock Movement**	**Output Vector**
Less than -2.0σ	Significant Drop	[1.0, 0, 0, 0, 0]
Greater than or equal to -2σ Less than -0.5σ	Moderate Drop	[0, 1.0, 0, 0, 0]
Greater than or equal to -0.5σ Less than 0.5σ	Stable	[0, 0, 1.0, 0, 0]
Greater than or equal to 0.5σ Less than 2.0σ	Moderate Increase	[0, 0, 0, 1.0, 0]
Greater than or equal to 2.0σ	Significant Increase	[0, 0, 0, 0, 1.0]

Table 4: Transforming Regression-based output to Classification

5.4.4 Training vs Testing Datasets

After datasets are generated for a forecasted stock, 95% of datasets are used for training a neural network while the remaining 5% act as testing data. If the actual performance of stock forecasting is to be measured, then datasets with

the latest pivot dates would be used for testing, as visually illustrated in Figure 18. However, this approach is not suitable for this research.

First aspect to consider is the underlying assumption that past price movement of 40 stocks can be used to forecast future movement of one stock. The stock selection process described in Section 5.4.1 is not sophisticated enough to provide confidence of such an assumption in realistic scenarios. The assumption is further challenged by the common notion in finance industry "past results are no indication of future" (Kennion, 2020).

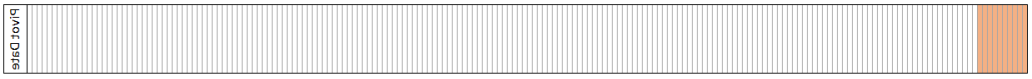

Figure 18: Select latest 5% of datasets for testing

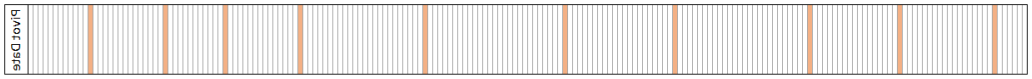

Figure 19: Randomly select 5% of datasets for testing

It is also important to recognize that the purpose of this research is to benchmark how well different neural networks *model* the constructed datasets instead of measuring the feasibility of forecasting stock market. Therefore, the method of selecting testing datasets shown in Figure 18 is not used.

Figure 19 illustrates the methodology of randomly selecting 5% of datasets for testing. With this approach, for each testing dataset, there are likely training datasets with pivot dates that are "close neighbors" to the pivot date of the testing dataset. This ensures that there are likely training datasets similar but not identical to testing datasets. Therefore, the resulting training-testing dataset combination is suitable to benchmark how well a neural network can model the

datasets without relying on the underlying assumption about stock market (stated in Section 5.2) being true.

134 different stocks are selected to be forecast. Their stock codes are listed in Appendix A. On average, over 6700 datasets are generated for each stock. There are 45,193 testing datasets in total, approximately 337 for each selected stock.

5.4.5 Implementation

The process of calling SQL store procedure, generating datasets, and selecting testing datasets are implemented in Visual Studio C#. For each forecasted stock, the resulting training and testing datasets are generated with the C# application, then passed to a C++ application via shared memory. The C++ application builds different neural networks using the framework described in Section 3.5 to train and test with the provided datasets.

6 Test

6.1 Setup

Like the preliminary test, three neural networks with different structures are constructed to benchmark the performance with the same set of datasets.

Dense-Only Neural Network		
Layer	**Parameters**	**Output Size**
--	--	$120 \times 40 = 4{,}800$
Dense	$4800 \times 256 = 1{,}228{,}800$	256
Dense	$256 \times 5 = 1{,}280$	5

CNN Neural Network		
Layer	**Parameters**	**Output Size**
--	--	$120 \times 40 = 4{,}800$
Convolutional Layer	10 channels of 19×1 filters $19 \times 1 \times 10 = 190$ parameters	$102 \times 40 \times 10 = 40{,}800$
Max Pooling Layer	Pooling by 3×1 No parameters	$34 \times 40 \times 10 = 13{,}600$
Convolutional Layer	10 channels of 20×1 filters $20 \times 1 \times 10 = 1{,}000$ parameters	$15 \times 40 \times 10 = 6{,}000$
Max Pooling Layer	Pooling by 3×1 No parameters	$5 \times 40 \times 10 = 2{,}000$
Dense	$2{,}000 \times 256 = 512{,}000$	256
Dense	$256 \times 5 = 1{,}280$	5

PCNN Neural Network		
Layer	**Parameters**	**Output Size**
--	--	$120 \times 40 = 4{,}800$
Partitioned Convolutional Layer	10 channels of 40 filters with size of 19×1 $19 \times 40 \times 10 = 7{,}600$ parameters	$102 \times 40 \times 10 = 40{,}800$
Max Pooling Layer	Pooling by 3×1 No parameters	$34 \times 40 \times 10 = 13{,}600$
Partitioned Convolutional Layer	10 channels of 40 filters with size of 20×1 $20 \times 40 \times 10 = 8{,}000$ parameters	$15 \times 40 \times 10 = 6{,}000$
Max Pooling Layer	Pooling by 3×1 No parameters	$5 \times 40 \times 10 = 2{,}000$
Dense	$2{,}000 \times 256 = 512{,}000$	256
Dense	$256 \times 5 = 1{,}280$	5

Figure 20 illustrates the structure of input data of a dataset. All data entries are of the same type (daily stock price change). But since each vector row represents different stock, the data cannot be processed with cross-row filter. Therefore, vector filter must be used.

$$\begin{bmatrix} i_{stock_0,0} & i_{stock_0,1} & i_{stock_0,2} & i_{stock_0,3} & \cdots & i_{stock_0,119} \\ i_{stock_1,0} & i_{stock_1,1} & i_{stock_1,2} & i_{stock_1,3} & \cdots & i_{stock_1,119} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ i_{stock_{39},0} & i_{stock_{39},1} & i_{stock_{39},2} & i_{stock_{39},3} & \cdots & i_{stock_{39},119} \end{bmatrix}$$

Figure 20: Structure of input data for CNN and PCNN

For CNN and PCNN, the output after each layer would have the same size, which is illustrated in Figure 21. However, all vector rows are processed by the same filter (for each channel) in CNN. For PCNN, each vector row is processed by unique filter. Therefore, PCNN has 7,600 trainable parameters for the first convolutional layer, 40 times of its CNN counterpart. Pooling layers have the same logic for both CNN and PCNN.

$$\begin{bmatrix} o_{stock_0,0} & o_{stock_0,1} & o_{stock_0,2} & o_{stock_0,3} & \cdots & o_{stock_0,101} \\ o_{stock_1,0} & o_{stock_1,1} & o_{stock_1,2} & o_{stock_1,3} & \cdots & o_{stock_1,101} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ o_{stock_{39},0} & o_{stock_{39},1} & o_{stock_{39},2} & o_{stock_{39},3} & \cdots & o_{stock_{39},101} \end{bmatrix}$$

Figure 21: Output of first convolutional layer for CNN and PCNN

The key difference between CNN and PCNN is the use of common vs. unique filter for each vector row. While CNN process the input matrix through a series of layers, PCNN partitions input matrix into separate vectors and process each in isolated series of convolutional and pooling layers until they are merged for final dense layers. Intuitively, CNN would be trained to learn the market-wise trends and patterns between the input data (representing the price changes of 40 stocks over 120 days) and the output (representing the 30-day price change of the forecasted stock). PCNN would be trained to identify the trends and patterns between each stock represented in the input data and the forecasted stock individually.

The weights and biases for all networks are initialized with the same method as described in Section 4.1. Every network is trained for 10 epochs, with mini-batch size of 64 and ReLU activation function (except final layer which uses Sigmoid). Test datasets are processed after each epoch to measure how well the network has been trained.

6.2 Initial Test Result

For every network, after each training epoch, the performance was measured with the test datasets. The best performance of the network and the number of training epochs to reach it were recorded. If performance drop with further training, the training would be terminated early before 10 epochs are completed. There are 134 sets of datasets, 45,193 testing datasets (averaging 337 per selected stock). The detailed results are recorded in Appendix A. Table 5 shows the summary of performance of each neural network structure.

Network Type	Only Dense	CNN	PCNN
Number of trainable parameters in network	1,230,080	513,670	528,880
Average number epochs to peak performance	2.31	5.81	8.31
Number of correct test data outputs (of 45,193 datasets)	27606	34253	37454
Accuracy	*61.08%*	*75.79%*	*82.88%*

Table 5: Forecast Accuracy by Network of Initial Test Run

Based on the result of the initial test run, the performance of PCNN seemed very promising, ranking top in accuracy. It is also observed that a dense-only neural network is not able to adequately model the data. However, before declaring PCNN as the superior neural network structure, it is important to ensure that the comparison was fair.

6.3 Additional Pre-processing

As explained in Section 5.4, datasets must be preprocessed to make the input data suitable for neural network modelling. One of the preprocesses is to convert raw stock price into percentages that represent daily change of stock prices. This is done so the parameter initialization for neural network is easier and the different average price between stocks does not cause issues.

However, even the volatility of the stock daily change, when expressed in percentage, still presents an issue when modelling with standard CNN. Two stocks can have correlation coefficient close to 1.0 but with drastic different volatility in price changes. Interestingly, this is not an issue for PCNN because every stock was processed with individual sets of layers and associated parameters. But for standard CNN, the same parameters are used to process data of all rows. This means input rows with larger volatility will "overshadow" the other rows as trainable parameters are updated to compensate for their higher standard deviation. Thus, for the initial test run, only a few rows from the input matrix contribute to the CNN model. This means PCNN had an "unfair" advantage because input data needs another preprocessing step to be suitable for standard CNN.

Original Data	**Standard Deviation**	**Scaled Data**
$\begin{bmatrix} -0.934 & -0.702 & -0.262 & 0.611 & 0.553 \\ -0.563 & 0.712 & 0.351 & 0.244 & -0.492 \\ -0.359 & -0.058 & 0.395 & -0.097 & -0.204 \\ -0.051 & 0.065 & 0.086 & -0.010 & 0.070 \\ -0.007 & -0.009 & 0.006 & -0.009 & -0.010 \end{bmatrix}$	$\begin{bmatrix} 0.708 \\ 0.556 \\ 0.282 \\ 0.059 \\ 0.007 \end{bmatrix}$	$\begin{bmatrix} \frac{-0.934}{0.708} \times 0.2 = -0.264 & -0.198 & -0.074 & 0.173 & 0.156 \\ -0.203 & 0.256 & 0.126 & 0.088 & -0.177 \\ -0255 & -0.041 & 0.280 & -0.069 & -0.144 \\ -0.171 & 0.219 & 0.289 & -0.035 & 0.236 \\ -0.214 & -0.262 & 0.185 & -0.256 & -0.297 \end{bmatrix}$

Figure 22: Scaling Data

The different volatility of movement between stocks can be mitigated by scaling (dividing) all daily price changes of a stock with its standard deviation. The result is then scaled down by a factor of 5.0 to keep the resulting values between 0 and 1. This process is demonstrated in Figure 22, which brings the standard deviation of every row to 0.2.

After scaling, the volatility of all stocks is brought to the same level, as illustrated in Figure 23. With input data further refined, test was executed again for standard CNN. It was not necessary to repeat the test for dense-only network and PCNN as the lack of scaling does not impact these two network structures.

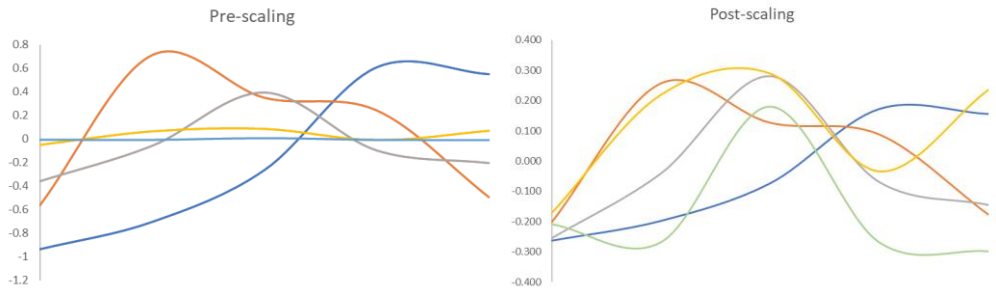

Figure 23: Impact of scaling to the volatility difference

6.4 Final Test Result

Table 6 shows the overall result with second round of testing for CNN. Standard CNN surpassed PCNN in performance and achieved the best accuracy. The detailed data of the test result is included in Appendix A.

Network Type	Only Dense	CNN	PCNN
Number of trainable parameters in network	1,230,080	513,670	528,880
Average number epochs to peak performance	2.31	6.37	8.31
Number of correct test data outputs (of 45,193 sets)	27,606	38,273	37,454
Accuracy	*61.08%*	*84.69%*	*82.88%*

Table 6: Forecast Accuracy by Network of Final Test Run

7 Discussion

With scaled input data, standard CNN has the best performance of all three networks. PCNN is a close second with 2.14% less accuracy. The dense-only network fell behind by a large margin.

7.1 Dense-only Network

Detailed data in Appendix A shows that the dense-only network merely "votes for majority". The neural network gravitates toward the most frequent appearing output in the training data and trains itself to always output that. The trained network would output the same result regardless of the input. This indicates an inability to identify complex patterns. This is also evidenced by how the network performance stop improving after only 2-3 training epochs.

7.2 Standard CNN vs PCNN

Both standard CNN and PCNN produced correct outputs for over 80% of test data. While the overall performance of CNN and PCNN were close, several observations can be made based on the detailed result data.

As described in Section 6.1, standard CNN uses underlying market-level trend to make the forecast while PCNN identifies the trends of 40 stocks individually. The outcome indicates that both methods make sense intuitively and lead to decent results. This is significant as it demonstrates a scenario where the proposed approach of partitioning input data with PCNN can produce performance that rivals standard CNN. Even though CNN outperformed PCNN by

a small margin, the result from PCNN indicates that this approach should not be overlooked for suitable datasets.

While the overall test data accuracies of CNN and PCNN are close, the difference in performance for some sets of datasets is quite large. This is likely related to underlying financial principles and nature of the stocks represented by the datasets. For PCNN, if one or several stocks represented by the input matrix have underlying correlation with the forecasted stock, its performance would likely be good. However, if such stocks are not included in the input data or simply do not exist, CNN would likely get better result with market-level trend. The method used to build input data (described in Section 5.4) is quite primitive. It can be assumed that PCNN would likely get a better overall result if a more sophisticated method (ideally based on financial principles) is used to select stocks for input data building. This also highlights that PCNN achieved its performance by identifying patterns different to those learned by CNN.

It is worth noting that there are also several potential factors that could have improved the performance of PCNN. CNN has a lower number of trainable parameters than PCNN. Although the difference of total number of trainable parameters between CNN and PCNN seems small (513,670 to 528,880, which is only different by 2.96%), the additions of parameters were all in the convolutional layers, raising the number of parameters in the first layer from 190 to 7,600 and second layer from 200 to 8,000. Increasing parameters by a factor of 40 implies that the layers would likely need more training data. It would also require more

training epochs to reach peak performance. This is shown by the test result where CNN takes an average of 5.4 training epochs to reach best result while PCNN takes an average of 7.3, as shown in Table 7. This indicates that the minor performance gap between PCNN and CNN may narrow or even reverse if the maximum number of training epochs was increased to 15. PCNN would likely benefit more than CNN does if there are more training data available.

Network Type	**CNN**	**PCNN**
Best Performance After 1 Epoch	8	3
Best Performance After 2 Epoch	8	1
Best Performance After 3 Epoch	10	3
Best Performance After 4 Epoch	11	2
Best Performance After 5 Epoch	15	3
Best Performance After 6 Epoch	12	6
Best Performance After 7 Epoch	17	12
Best Performance After 8 Epoch	12	24
Best Performance After 9 Epoch	16	36
Best Performance After 10 Epoch	25	44
Average Epochs to Best Performance	6.373	8.306

Table 7: Number of Epochs to Reach Best Performance

7.3 Viability of PCNN

Test results have indicated that PCNN has potentially the same capability of modeling data as that of standard CNN. Its performance would likely benefit under the following conditions:

- Sophisticated method to construct the input groups (if selection is required)
- Large amount of training datasets
- High number of training epochs

The test conducted in this research shows a scenario where both CNN and PCNN are viable options. This means that PCNN can be a replacement for standard CNN in certain situations or a supplement to standard CNN to create a more powerful neural network (discussed in section below). For the scope of this research, the result of the test has adequately demonstrated the viability of PCNN.

7.4 Application of PCNN

From a structural point of view, PCNN is designed to handle input with heterogenous groups of data. However, even without the concept of PCNN, data analysist would still construct the network to convolute these data groups separately, which leads the same effect of PCNN. A particularly interesting possibility explored by this research is using PCNN even when the input data is also suitable for standard CNN.

Processing homogenous groups of data with separate convolutional layers may seem counterintuitive. But the test result in this research demonstrates that identifying trends and patterns for individual data groups could produce comparable or even superior results. One does not need to choose between PCNN and standard CNN when constructing a neural network. Data can be processed with both methods and brought together through dense layers or a voting mechanism. Figure 24 shows such an example. Note that PCNN part and CNN part of the network can have different number of layers and filter dimensions.

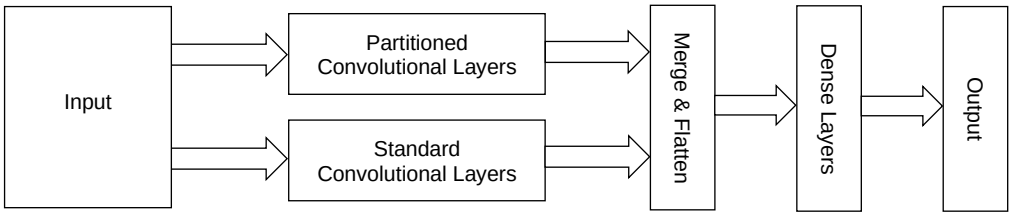

Figure 24: Example of Flexible PCNN Design

8 Future Work

The next step is to further explore the application of PCNN with datasets that are normally processed with standard CNN, like the stock data used for this research. If the performance of PCNN proves to be good with the datasets, further research could be conducted to determine if an even better result can be achieved by pairing standard CNN and PCNN, as described in Section 7.4.

9 Conclusion

This research sets forth an alternative to the standard convolutional neural network structure when modelling homogenous groups of data. The proposed PCNN is designed to identify underlying patterns of individual data groups that are not learned by standard CNN. The author of this research implemented the software framework that allows both CNN and PCNN to be constructed and test. In addition, test data was built from stock markets and refined to be suitable for both CNN and PCNN. The test results show that PCNN is capable of meaningful performance when modelling suitable datasets. Post-test analysis indicates that there are possible methods to improve its performance even further. The result

indicates that with suitable datasets and right setups, PCNN can potentially be used to replace or pair with standard CNN for better neural network performance.

10 References

Alpha Vantage Inc. (2019). *Alpha Vantage - Free APIs for Realtime and Historical Stock*. (Alpha Vantage Inc.) Retrieved March 3, 2018, from Alpha Vantage: https://www.alphavantage.co/

Brownlee, J. (2017, July 21). *A Gentle Introduction to Mini-Batch Gradient Descent and How to Configure Batch Size*. Retrieved February 28, 2019, from Machine Learning Mastery: https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/

Brownlee, J. (2017, December 11). *Difference Between Classification and Regression in Machine Learning.* Retrieved June 13, 2019, from Machine Learning Mastery: https://machinelearningmastery.com/classification-versus-regression-in-machine-learning/

Brownlee, J. (2017, July 3). *Gentle Introduction to the Adam Optimization Algorithm for Deep Learning*. Retrieved May 2, 2019, from Machine Learning Mastery: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

Brownlee, J. (2019, January 18). *How to Accelerate Learning of Deep Neural Networks With Batch Normalization.* Retrieved March 11, 2019, from Machine Learning Mastery: https://machinelearningmastery.com/how-to-accelerate-learning-of-deep-neural-networks-with-batch-normalization/

EODData, LLC. (2019). *End of Day Stock Quote Data and Historical Stock Prices*. (EODData, LLC.) Retrieved Feb 28, 2018, from eoddata: http://www.eoddata.com/

Google Brain Team. (2020, July 9). *TensorFlow*. Retrieved Feb 28, 2019, from TensorFlow: https://www.tensorflow.org/

Harris, M. (2007). Optimizing Parallel Reduction in CUDA.

Harris, M. (2013, January 28). *Using Shared Memory in CUDA C/C++*. (NVIDIA Corporation) Retrieved June 3, 2018, from https://devblogs.nvidia.com/using-shared-memory-cuda-cc/

Hyungui Lim, J. P. (2017, November 16). Rare Sound Event Detection Using 1D Convolutional Recurrent Neural Networks. *Detection and Classification of Acoustic Scenes and Events 2017*.

Isaksson, M. (2020, June 6). *Four Common Types of Neural Network Layers*. Retrieved from Towards Data Science: https://towardsdatascience.com/four-common-types-of-neural-network-layers-c0d3bb2a966c

Kennion, J. (2020, December 08). *Past Performance Is No Guarantee of Future Results*. Retrieved from the balance: https://www.thebalance.com/past-performance-is-no-guarantee-of-future-results-357862

Kim, Y. (2014). *Convolutional Neural Networks for Sentence Classification.* New York: New York University.

Massachusetts Institute of Technology. (2020). *Keras: the Python deep learning API*. Retrieved February 28, 2019, from Keras: https://keras.io/

Nielsen, M. A. (2015, June). *Neural Networks and Deep Learning.* Determination Press. Retrieved August 31, 2019

NVIDIA Corporation. (2013, July). Tesla K20X GPU Accelerator.

NVIDIA Corporation. (2017, June 2). *CUDA Occupancy Calculator :: CUDA Toolkit Documentation.* Retrieved June 3, 2018, from https://docs.nvidia.com/cuda/cuda-occupancy-calculator/index.html

NVIDIA Corporation. (2017, July 2). *CUDA Toolkit Documentation*. Retrieved November 11, 2018, from https://docs.nvidia.com/cuda/

NVIDIA Corporation. (2019). *GeForce GTX 1050 Graphics Cards | NVIDIA GeForce*. Retrieved from NVIDIA: https://www.nvidia.com/en-sg/geforce/products/10series/geforce-gtx-1050/

NVIDIA Corporation. (n.d.). *Whitepaper NVidia's Next Generation CUDA Compute Architecture: Fermi.* Retrieved July 9, 2018, from https://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf

Osama Abdeljaber, O. A. (2017, February 3). Real-time Vibration-based Structural Damage Detection Using One-dimensional Convolutional Neural Networks. *Journal of Sound and Vibration, 388*, 154-170.

Serkan Kiranyaz, O. A. (2019, May 9). *1D Convolutional Neural Networks and Applications: A Survey.* Retrieved from arXiv.org: https://arxiv.org/ftp/arxiv/papers/1905/1905.03554.pdf

Siavash Sakhavi, C. G. (2015). Parallel Convolutional-linear Neural Network for Motor Imagery Classification. *2015 23rd European Signal Processing Conference (EUSIPCO).* Nice, France: IEEE.

Torch Contributors. (2020). *PyTorch documentation — PyTorch 1.6.0 documentation*. Retrieved February 28, 2019, from PyTorch: https://pytorch.org/

Turker Ince, S. K. (2016, November). Real-Time Motor Fault Detection by 1-D Convolutional Neural Networks. *IEEE Transactions on Industrial Electronics, 63*(11), 7067-7075.

Woolley, C. (2013). GPU Optimization Fundamentals. NVIDIA.

Yahoo! (2019). *Yahoo Finance - Stock Market Live, Quotes, Business & Financial News*. (Yahoo!) Retrieved March 3, 2018, from Yahoo Finance: https://finance.yahoo.com/

Yakura, H. (March 2018). Malware Analysis of Imaged Binary Samples by Convolutional Neural Network with Attention Mechanism. *The 8th ACM Conference on Data and Application Security and Privacy* (pp. 127-134). Tempe, AZ, USA: ACM CODASPY '18.

Yann LeCun, C. C. (2020). *The MNIST database of handwritten digits*. Retrieved January 15, 2018, from The MNIST database: http://yann.lecun.com/exdb/mnist/

Appendix A Test Result Details

Datasets are built for 134 stocks. The performance of each test neural network is shown in this section. The following columns are included for each table:

- # of Epochs to Best: the number of training epochs before peak performance of the neural network is reached
- The network has five possible outputs, as described in Section 5.4. There are two columns for each possible output:
    - Total: number of test datasets with such expected result
    - Corr.: number of test datasets for which the network correctly forecasted such output
- Incorrect – Magnitude of Error: number of incorrect outputs categorized by how incorrect they are. For example:
    - An actual output of $[0, 1, 0, 0, 0]$ against an expected output of $[0, 0, 1, 0, 0]$ means an error magnitude of 1
    - An actual output of $[0, 1, 0, 0, 0]$ against an expected output of $[0, 0, 0, 0, 1]$ means an error magnitude of 3

A.1 Dense-only Neural Network

Code	# of Epochs to Best	Significant Drop		Moderate Drop		Stable		Moderate Increase		Significant Increase		Incorrect – Magnitude of Error			
		Total	Corr.	Total	Corr.	Total	Corr.	Total	Corr.	Total	Corr.	1	2	3	4
ABB	2	0	0	21	0	26	0	225	222	43	5	67	21	0	0
AKS	1	29	0	213	213	45	0	24	0	4	0	74	24	4	0
BIOS	1	303	303	3	0	7	0	2	0	0	0	3	7	2	0
CIA	1	0	0	14	0	44	0	179	179	78	0	122	14	0	0
COT	1	0	0	39	0	264	264	5	0	7	0	44	7	0	0
CYBE	1	2	0	77	0	160	160	57	0	19	0	134	21	0	0
DYNT	1	0	0	105	0	207	207	61	0	15	0	166	15	0	0
EFOI	1	0	0	8	0	348	348	30	0	2	0	38	2	0	0
EMITF	1	30	0	189	189	99	1	35	0	3	0	128	35	3	0
FISV	1	68	0	235	235	69	0	13	0	3	0	137	13	3	0
GEOS	2	0	0	21	0	90	9	87	87	15	0	101	16	0	0
GFI	2	0	0	6	0	101	7	224	218	57	0	154	9	0	0
GILT	1	2	0	71	0	199	166	117	50	18	0	177	14	0	0
GPIC	1	0	0	127	0	158	158	20	0	10	0	147	10	0	0
GPX	2	0	0	0	0	12	0	77	6	267	264	76	10	0	0
GV	6	0	0	0	0	78	9	250	237	28	6	100	4	0	0
HDSN	1	0	0	45	0	199	199	90	0	22	0	135	22	0	0
HLIT	5	0	0	16	0	117	34	176	162	47	5	138	17	0	0
HMY	1	0	0	173	168	93	10	4	0	6	0	90	4	4	0
HOV	9	3	0	130	15	181	130	74	39	19	0	205	18	0	0
HRTX	2	47	4	202	193	52	10	10	0	4	0	94	10	4	0
ICAD	1	35	0	278	278	65	0	20	0	9	0	100	20	9	0
IDRA	1	3	0	341	341	50	0	15	0	4	0	53	15	4	0
IDSA	1	0	0	4	0	281	148	111	74	11	0	185	0	0	0
IESC	1	79	0	249	249	52	0	29	0	4	0	131	29	4	0
IGLD	7	0	0	12	2	157	45	114	90	32	13	146	18	1	0
IGLD	1	276	276	0	0	0	0	0	0	0	0	0	0	0	0
IIN	1	0	0	125	0	197	196	68	0	17	0	194	17	0	0
IMGN	8	0	0	2	0	134	29	192	177	28	2	147	1	0	0
IMGN	5	0	0	0	0	101	24	270	267	36	1	115	0	0	0
IMH	3	0	0	22	0	329	325	45	6	17	0	65	17	0	0
IMMU	3	0	0	0	0	325	325	28	3	3	0	25	3	0	0
ING	8	0	0	6	0	21	0	206	151	123	64	133	8	0	0
INOD	10	2	0	106	67	117	41	36	7	15	6	116	35	4	0
INS	2	5	0	112	27	152	127	35	0	11	0	139	21	1	0

Code	# of Epochs to Best	Significant Drop		Moderate Drop		Stable		Moderate Increase		Significant Increase		Incorrect – Magnitude of Error			
		Total	Corr.	Total	Corr.	Total	Corr.	Total	Corr.	Total	Corr.	1	2	3	4
INSG	2	79	29	99	87	22	0	13	0	0	0	83	13	1	0
IPAS	4	109	12	87	82	12	0	5	0	0	0	114	5	0	0
ISIG	1	0	0	62	0	177	177	60	0	16	0	120	18	0	0
JAKK	5	5	0	80	8	70	7	53	49	5	0	87	61	1	0
JCS	1	1	0	84	1	187	131	117	60	18	0	183	32	0	0
JOB	1	0	0	77	0	266	266	62	0	8	0	139	8	0	0
JPM	6	9	0	83	13	185	123	117	44	13	0	195	32	0	0
KGC	2	93	39	111	92	7	0	1	0	1	0	80	1	1	0
KGJI	1	0	0	0	0	279	279	28	0	8	0	28	8	0	0
KIQ	1	0	0	44	0	276	234	70	7	17	0	141	25	0	0
KMT	1	0	0	0	0	4	0	32	0	320	320	32	4	0	0
KVHI	1	254	254	20	0	2	0	0	0	0	0	20	2	0	0
LEE	1	0	0	74	0	289	289	40	0	10	0	114	10	0	0
LFVN	1	0	0	40	0	241	241	62	0	13	0	102	13	0	0
LPX	1	23	0	243	238	102	18	34	0	11	0	121	35	1	0
LSCC	1	7	0	103	6	162	156	73	3	11	0	169	22	0	0
LTBR	1	0	0	16	0	295	295	2	0	2	0	18	2	0	0
LYTS	1	46	0	168	168	51	0	8	0	3	0	97	8	3	0
MAGS	1	10	0	245	245	40	0	16	0	4	0	50	16	4	0
MAMS	1	0	0	6	0	73	0	180	180	17	0	90	6	0	0
MERC	3	15	0	252	179	122	66	14	1	10	0	148	19	0	0
MFIN	4	27	0	188	159	139	34	22	0	12	0	164	30	1	0
MICR	8	0	0	15	0	251	126	123	76	18	1	175	29	0	0
MIND	5	0	0	12	0	74	7	246	218	81	25	146	17	0	0
MNI	2	0	0	83	4	197	196	24	0	11	0	104	11	0	0
MNTA	6	0	0	64	22	106	85	34	10	9	0	86	9	1	0
MSTR	1	0	0	30	0	254	251	64	1	8	0	96	8	0	0
MXC	7	0	0	104	6	283	280	14	0	6	0	107	11	3	0
MXWL	1	0	0	20	0	76	0	228	228	64	0	140	20	0	0
NAII	1	0	0	29	0	136	0	185	185	38	0	174	29	0	0
NSYS	1	8	0	64	0	156	156	70	0	17	0	134	25	0	0
NTZ	3	0	0	17	0	98	12	128	127	33	0	119	18	0	0
NVLN	1	145	13	153	150	11	0	6	0	0	0	146	6	0	0
NYMX	1	0	0	64	0	255	255	87	0	7	0	151	7	0	0
NYNY	4	0	0	19	0	170	115	118	78	8	0	114	7	1	0
OI	1	260	260	15	0	0	0	1	0	0	0	15	0	1	0
OSUR	2	0	0	24	0	158	33	163	142	43	1	168	43	1	0

Code	# of Epochs to Best	Significant Drop		Moderate Drop		Stable		Moderate Increase		Significant Increase		Incorrect – Magnitude of Error			
		Total	Corr.	Total	Corr.	Total	Corr.	Total	Corr.	Total	Corr.	1	2	3	4
PAR	7	103	60	140	98	25	3	4	0	4	0	103	10	2	0
PDLI	1	0	0	43	0	246	245	61	4	6	0	101	6	0	0
PESI	1	0	0	5	0	30	0	131	131	47	0	77	5	0	0
PICO	5	0	0	0	0	11	0	94	15	308	299	91	8	0	0
PIR	1	0	0	19	0	241	241	11	0	5	0	30	5	0	0
PKD	1	96	1	225	225	72	0	9	0	5	0	167	9	5	0
PLX	1	0	0	0	0	264	264	7	0	5	0	7	5	0	0
PPIH	3	17	0	265	205	92	29	25	0	8	0	148	18	7	0
PRAN	2	1	0	63	4	101	101	40	0	8	0	99	9	0	0
PRCP	4	0	0	0	0	3	0	93	10	260	258	85	3	0	0
QCOM	1	0	0	26	0	250	116	118	97	19	0	178	22	0	0
QUMU	2	5	0	81	5	87	85	34	2	6	0	110	11	0	0
RADA	1	0	0	45	0	241	240	90	0	31	0	136	31	0	0
RAND	1	0	0	107	64	86	44	13	0	7	0	92	10	3	0
RAVE	1	6	0	177	67	168	123	52	0	10	0	198	24	1	0
RCMT	6	223	157	164	81	17	3	9	0	0	0	153	14	5	0
RELL	1	56	0	182	182	61	0	13	0	3	0	117	13	3	0
RLH	1	1	0	5	0	55	0	130	130	22	0	77	5	1	0
SCKT	1	0	0	83	0	146	146	39	0	8	0	122	8	0	0
SCX	1	15	0	123	2	179	178	80	0	16	0	202	31	0	0
SFE	8	0	0	5	0	82	48	230	161	39	25	98	23	1	0
SGMA	2	0	0	20	0	268	201	106	61	19	0	139	12	0	0
SGU	1	11	0	46	0	179	140	109	29	11	0	153	27	7	0
SIGA	1	0	0	39	0	258	252	97	5	19	0	141	15	0	0
SIGM	1	0	0	80	0	212	212	91	0	24	0	171	24	0	0
SMRT	1	0	0	3	0	167	167	30	0	13	0	33	13	0	0
SPB	1	1	0	11	0	61	0	235	235	48	0	109	11	1	0
SRDX	3	13	0	117	109	64	13	14	0	5	0	77	14	0	0
SRI	1	4	0	67	0	139	139	54	0	12	0	121	16	0	0
STAR	1	5	0	33	0	199	199	22	0	17	0	55	22	0	0
STRM	1	0	0	0	0	2	0	252	104	102	62	188	2	0	0
SYPR	6	0	0	10	0	201	42	139	135	38	1	205	5	0	0
TACT	1	0	0	78	0	197	145	119	37	19	0	219	12	0	0
TCI	1	3	0	80	0	191	191	90	0	24	0	170	27	0	0
THC	6	10	0	162	19	182	169	44	1	15	2	201	19	1	1
TIVO	1	28	0	132	132	41	0	11	0	1	0	69	11	1	0
TRK	1	0	0	0	0	10	0	117	0	229	229	117	10	0	0

Code	# of Epochs to Best	Significant Drop		Moderate Drop		Stable		Moderate Increase		Significant Increase		Incorrect – Magnitude of Error			
		Total	Corr.	Total	Corr.	Total	Corr.	Total	Corr.	Total	Corr.	1	2	3	4
TRT	8	0	0	70	7	161	71	124	64	33	7	193	41	5	0
TTMI	1	312	312	1	0	2	0	0	0	0	0	1	2	0	0
TWMC	1	65	0	248	248	59	0	24	0	11	0	124	24	11	0
UAMY	1	0	0	174	174	31	0	4	0	4	0	31	4	4	0
UCTT	4	0	0	9	0	87	25	99	88	18	10	76	11	3	0
UIS	2	1	0	148	12	148	129	83	20	8	0	213	14	0	0
UNM	1	0	0	14	0	25	0	217	131	100	60	138	17	10	0
USAK	1	79	0	250	250	50	0	20	0	8	0	129	20	8	0
USAS	1	18	0	162	162	27	0	3	0	3	0	45	3	3	0
USAU	1	70	0	166	166	24	0	13	0	3	0	94	13	3	0
USEG	2	0	0	114	14	190	170	90	15	19	0	194	20	0	0
USG	2	0	0	26	0	135	12	217	210	29	0	161	24	0	0
VGZ	1	10	0	154	154	34	0	12	0	3	0	44	12	3	0
VICR	2	126	25	209	199	54	0	16	0	2	0	164	17	2	0
VIRC	1	336	336	44	0	4	0	4	0	0	0	44	4	4	0
VOXX	1	0	0	62	0	224	149	99	55	28	0	188	21	0	0
VTNR	1	0	0	0	0	100	14	99	93	14	0	106	0	0	0
VVUS	1	0	0	153	19	162	161	81	0	11	0	216	11	0	0
WLK	1	197	197	15	0	1	0	0	0	0	0	15	1	0	0
WTT	1	0	0	127	1	209	208	46	0	6	0	173	6	0	0
WTW	9	0	0	24	5	197	189	43	9	12	1	65	7	0	0
WYY	1	165	165	34	0	6	0	6	0	2	0	34	6	6	2
X	1	31	0	205	192	105	10	37	0	10	0	143	35	8	0
XEC	3	14	0	126	41	100	68	34	13	2	0	131	21	2	0
ZIXI	3	0	0	4	0	157	65	197	172	30	0	148	3	0	0

A.2 Standard Convolutional Neural Network

Code	# of Epochs to Best	Significant Drop		Moderate Drop		Stable		Moderate Increase		Significant Increase		Incorrect – Magnitude of Error			
		Total	Corr.	Total	Corr.	Total	Corr.	Total	Corr.	Total	Corr.	1	2	3	4
ABB	2	0	0	21	6	26	1	225	218	43	16	57	17	0	0
AKS	1	29	3	213	158	45	21	24	10	4	0	81	30	12	0
BIOS	5	303	303	3	1	7	6	2	1	0	0	3	1	0	0
CIA	8	0	0	14	10	44	15	179	150	78	65	60	12	3	0
COT	5	0	0	39	33	264	264	5	0	7	5	11	2	0	0
CYBE	7	2	0	77	34	160	105	57	52	19	8	102	14	0	0
DYNT	4	0	0	105	48	207	192	61	24	15	12	97	14	1	0
EFOI	1	0	0	8	0	348	340	30	2	2	0	44	2	0	0
EMITF	1	30	0	189	170	99	45	35	11	3	0	112	17	1	0
FISV	9	68	53	235	216	69	26	13	9	3	0	76	6	1	1
GEOS	7	0	0	21	10	90	66	87	76	15	14	38	6	3	0
GFI	2	0	0	6	3	101	41	224	211	57	27	103	3	0	0
GILT	2	2	0	71	37	199	192	117	52	18	10	104	12	0	0
GPIC	6	0	0	127	113	158	114	20	9	10	0	67	8	4	0
GPX	3	0	0	0	0	12	5	77	21	267	261	67	2	0	0
GV	10	0	0	0	0	78	23	250	243	28	21	66	3	0	0
HDSN	7	0	0	45	15	199	174	90	47	22	16	83	19	2	0
HLIT	4	0	0	16	1	117	100	176	153	47	27	57	17	1	0
HMY	6	0	0	173	154	93	77	4	0	6	6	38	1	0	0
HOV	7	3	0	130	47	181	158	74	51	19	14	120	16	1	0
HRTX	5	47	23	202	185	52	32	10	7	4	3	64	1	0	0
ICAD	7	35	8	278	268	65	17	20	15	9	3	81	12	3	0
IDRA	7	3	1	341	311	50	28	15	8	4	2	48	5	9	1
IDSA	5	0	0	4	0	281	139	111	92	11	10	153	13	0	0
IESC	5	79	13	249	241	52	12	29	10	4	1	118	15	3	0
IGLD	10	0	0	12	9	157	120	114	68	32	29	73	15	1	0
IGLD	1	276	276	0	0	0	0	0	0	0	0	0	0	0	0
IIN	2	0	0	125	48	197	141	68	55	17	11	108	38	6	0
IMGN	8	0	0	2	1	134	70	192	177	28	16	91	1	0	0
IMGN	3	0	0	0	0	101	48	270	261	36	7	91	0	0	0
IMH	9	0	0	22	2	329	324	45	10	17	13	54	6	4	0
IMMU	9	0	0	0	0	325	296	28	23	3	3	33	1	0	0
ING	3	0	0	6	2	21	4	206	197	123	77	72	4	0	0
INOD	10	2	0	106	90	117	98	36	18	15	9	45	12	4	0
INS	5	5	4	112	71	152	76	35	32	11	8	103	20	1	0

Code	# of Epochs to Best	Significant Drop		Moderate Drop		Stable		Moderate Increase		Significant Increase		Incorrect – Magnitude of Error			
		Total	Corr.	Total	Corr.	Total	Corr.	Total	Corr.	Total	Corr.	1	2	3	4
INSG	7	79	71	99	97	22	11	13	10	0	0	21	3	0	0
IPAS	7	109	98	87	67	12	7	5	3	0	0	37	1	0	0
ISIG	4	0	0	62	44	177	171	60	40	16	7	46	7	0	0
JAKK	7	5	3	80	58	70	30	53	50	5	0	52	19	1	0
JCS	2	1	0	84	18	187	151	117	74	18	8	130	24	2	0
JOB	4	0	0	77	27	266	233	62	45	8	7	96	5	0	0
JPM	5	9	0	83	40	185	121	117	97	13	8	116	20	3	2
KGC	4	93	74	111	88	7	0	1	0	1	1	48	2	0	0
KGJI	10	0	0	0	0	279	272	28	4	8	2	28	9	0	0
KIQ	1	0	0	44	1	276	203	70	29	17	1	147	26	0	0
KMT	5	0	0	0	0	4	0	32	5	320	320	27	4	0	0
KVHI	9	254	250	20	14	2	2	0	0	0	0	6	4	0	0
LEE	3	0	0	74	66	289	273	40	34	10	7	32	0	1	0
LFVN	10	0	0	40	19	241	187	62	57	13	1	78	14	0	0
LPX	8	23	0	243	216	102	64	34	33	11	6	80	6	8	0
LSCC	5	7	0	103	30	162	105	73	62	11	6	92	60	1	0
LTBR	4	0	0	16	5	295	294	2	0	2	1	14	1	0	0
LYTS	5	46	26	168	114	51	44	8	6	3	3	69	7	3	4
MAGS	9	10	4	245	238	40	26	16	8	4	2	27	8	2	0
MAMS	6	0	0	6	1	73	49	180	151	17	14	46	12	3	0
MERC	4	15	3	252	178	122	46	14	13	10	6	92	63	12	0
MFIN	10	27	23	188	159	139	124	22	8	12	8	60	5	1	0
MICR	4	0	0	15	7	251	186	123	98	18	12	97	7	0	0
MIND	9	0	0	12	0	74	28	246	223	81	42	107	13	0	0
MNI	10	0	0	83	68	197	190	24	19	11	6	31	1	0	0
MNTA	4	0	0	64	39	106	103	34	22	9	4	42	3	0	0
MSTR	4	0	0	30	4	254	241	64	47	8	0	54	10	0	0
MXC	2	0	0	104	50	283	270	14	2	6	6	77	2	0	0
MXWL	5	0	0	20	4	76	31	228	222	64	31	85	11	4	0
NAII	10	0	0	29	20	136	105	185	95	38	37	102	23	6	0
NSYS	7	8	6	64	11	156	95	70	63	17	14	85	29	12	0
NTZ	8	0	0	17	6	98	66	128	105	33	24	54	15	6	0
NVLN	2	145	120	153	125	11	8	6	5	0	0	51	3	3	0
NYMX	6	0	0	64	24	255	236	87	60	7	5	78	10	0	0
NYNY	5	0	0	19	1	170	155	118	89	8	3	56	10	1	0
OI	9	260	251	15	10	0	0	1	1	0	0	13	1	0	0
OSUR	5	0	0	24	14	158	116	163	132	43	26	77	19	4	0

Code	# of Epochs to Best	Significant Drop		Moderate Drop		Stable		Moderate Increase		Significant Increase		Incorrect – Magnitude of Error			
		Total	Corr.	Total	Corr.	Total	Corr.	Total	Corr.	Total	Corr.	1	2	3	4
PAR	7	103	56	140	126	25	16	4	0	4	0	72	4	2	0
PDLI	2	0	0	43	20	246	164	61	46	6	2	106	18	0	0
PESI	7	0	0	5	0	30	17	131	117	47	35	34	6	4	0
PICO	6	0	0	0	0	11	0	94	35	308	301	69	8	0	0
PIR	10	0	0	19	8	241	237	11	8	5	4	16	3	0	0
PKD	10	96	43	225	182	72	50	9	4	5	4	103	20	1	0
PLX	1	0	0	0	0	264	254	7	1	5	0	9	12	0	0
PPIH	8	17	3	265	219	92	53	25	8	8	6	97	18	3	0
PRAN	7	1	1	63	29	101	97	40	21	8	5	55	5	0	0
PRCP	4	0	0	0	0	3	0	93	40	260	253	60	3	0	0
QCOM	2	0	0	26	12	250	227	118	61	19	2	105	5	1	0
QUMU	10	5	4	81	72	87	64	34	14	6	5	44	10	0	0
RADA	9	0	0	45	27	241	225	90	64	31	30	58	3	0	0
RAND	6	0	0	107	98	86	55	13	3	7	6	49	1	1	0
RAVE	5	6	2	177	154	168	73	52	43	10	7	124	9	1	0
RCMT	7	223	190	164	97	17	5	9	4	0	0	103	11	3	0
RELL	9	56	8	182	167	61	54	13	3	3	1	57	19	6	0
RLH	8	1	0	5	3	55	39	130	120	22	18	31	0	1	1
SCKT	7	0	0	83	46	146	123	39	33	8	1	64	9	0	0
SCX	9	15	0	123	60	179	99	80	71	16	12	123	42	6	0
SFE	2	0	0	5	3	82	44	230	199	39	31	71	8	0	0
SGMA	9	0	0	20	7	268	244	106	73	19	10	50	25	4	0
SGU	10	11	3	46	27	179	142	109	69	11	7	84	18	5	1
SIGA	8	0	0	39	8	258	216	97	60	19	18	86	19	6	0
SIGM	4	0	0	80	54	212	206	91	37	24	0	84	24	2	0
SMRT	3	0	0	3	1	167	164	30	15	13	9	21	3	0	0
SPB	7	1	0	11	1	61	20	235	195	48	40	68	23	8	1
SRDX	8	13	2	117	108	64	56	14	10	5	5	31	1	0	0
SRI	10	4	0	67	16	139	127	54	35	12	3	81	14	0	0
STAR	7	5	0	33	8	199	173	22	17	17	15	34	17	7	5
STRM	9	0	0	0	0	2	0	252	106	102	94	154	2	0	0
SYPR	2	0	0	10	0	201	160	139	89	38	6	130	3	0	0
TACT	2	0	0	78	6	197	161	119	92	19	11	118	24	1	0
TCI	1	3	0	80	27	191	178	90	22	24	0	117	43	1	0
THC	7	10	7	162	96	182	177	44	27	15	14	88	3	1	0
TIVO	4	28	13	132	112	41	34	11	3	1	0	50	1	0	0
TRK	9	0	0	0	0	10	3	117	77	229	192	78	6	0	0

Code	# of Epochs to Best	Significant Drop		Moderate Drop		Stable		Moderate Increase		Significant Increase		Incorrect – Magnitude of Error			
		Total	Corr.	Total	Corr.	Total	Corr.	Total	Corr.	Total	Corr.	1	2	3	4
TRT	3	0	0	70	8	161	118	124	113	33	9	113	27	0	0
TTMI	1	312	312	1	0	2	0	0	0	0	0	1	2	0	0
TWMC	8	65	9	248	234	59	40	24	5	11	0	96	18	5	0
UAMY	2	0	0	174	167	31	19	4	3	4	0	20	0	4	0
UCTT	10	0	0	9	4	87	54	99	96	18	11	42	3	3	0
UIS	7	1	1	148	108	148	135	83	69	8	5	63	7	0	0
UNM	1	0	0	14	3	25	0	217	157	100	62	112	16	6	0
USAK	1	79	3	250	249	50	3	20	0	8	0	125	19	8	0
USAS	10	18	16	162	158	27	17	3	1	3	1	16	2	2	0
USAU	10	70	57	166	143	24	20	13	5	3	2	44	2	3	0
USEG	8	0	0	114	58	190	158	90	79	19	13	88	17	0	0
USG	9	0	0	26	3	135	103	217	194	29	22	83	2	0	0
VGZ	6	10	7	154	139	34	11	12	7	3	2	31	15	1	0
VICR	7	126	95	209	198	54	40	16	12	2	2	57	3	0	0
VIRC	1	336	336	44	3	4	1	4	0	0	0	42	5	1	0
VOXX	10	0	0	62	42	224	198	99	83	28	23	61	3	3	0
VTNR	5	0	0	0	0	100	75	99	92	14	7	38	1	0	0
VVUS	6	0	0	153	71	162	144	81	33	11	7	141	10	1	0
WLK	5	197	193	15	12	1	1	0	0	0	0	7	0	0	0
WTT	2	0	0	127	24	209	183	46	32	6	3	107	36	3	0
WTW	8	0	0	24	18	197	183	43	34	12	7	34	0	0	0
WYY	2	165	157	34	19	6	3	6	4	2	0	27	2	1	0
X	2	31	8	205	180	105	37	37	31	10	2	100	16	14	0
XEC	8	14	5	126	74	100	93	34	21	2	1	73	4	5	0
ZIXI	6	0	0	4	0	157	105	197	179	30	7	87	10	0	0

A.3 Standard Convolutional Neural Network with Data Scaling

Code	# of Epochs to Best	Significant Drop		Moderate Drop		Stable		Moderate Increase		Significant Increase		Incorrect – Magnitude of Error			
		Total	Corr.	Total	Corr.	Total	Corr.	Total	Corr.	Total	Corr.	1	2	3	4
ABB	10	0	0	21	9	26	12	225	223	43	25	36	10	0	0
AKS	9	29	24	213	164	45	41	24	22	4	4	57	1	2	0
BIOS	5	303	303	3	2	7	6	2	2	0	0	1	1	0	0
CIA	9	0	0	14	12	44	30	179	172	78	61	37	2	1	0
COT	9	0	0	39	36	264	263	5	2	7	7	7	0	0	0
CYBE	7	2	2	77	65	160	141	57	52	19	18	33	4	0	0
DYNT	2	0	0	105	44	207	204	61	32	15	11	90	5	2	0
EFOI	1	0	0	8	5	348	341	30	18	2	1	23	0	0	0
EMITF	8	30	23	189	136	99	86	35	25	3	0	84	2	0	0
FISV	9	68	59	235	222	69	39	13	10	3	0	58	0	0	0
GEOS	7	0	0	21	18	90	86	87	86	15	15	7	0	1	0
GFI	6	0	0	6	5	101	73	224	199	57	46	64	1	0	0
GILT	10	2	2	71	48	199	191	117	94	18	10	55	7	0	0
GPIC	5	0	0	127	107	158	144	20	7	10	9	47	1	0	0
GPX	7	0	0	0	0	12	9	77	57	267	262	27	1	0	0
GV	5	0	0	0	0	78	37	250	239	28	22	55	3	0	0
HDSN	9	0	0	45	16	199	183	90	65	22	17	63	11	1	0
HLIT	6	0	0	16	1	117	96	176	171	47	33	46	9	0	0
HMY	3	0	0	173	168	93	72	4	3	6	6	26	1	0	0
HOV	10	3	2	130	104	181	168	74	63	19	18	51	1	0	0
HRTX	8	47	30	202	185	52	37	10	7	4	2	51	2	1	0
ICAD	6	35	11	278	240	65	38	20	19	9	1	69	28	1	0
IDRA	6	3	1	341	321	50	43	15	9	4	4	34	0	1	0
IDSA	2	0	0	4	1	281	169	111	107	11	9	119	2	0	0
IESC	6	79	27	249	244	52	16	29	16	4	3	100	7	0	0
IGLD	1	276	276	0	0	0	0	0	0	0	0	0	0	0	0
IGLD	6	0	0	12	9	157	128	114	88	32	29	55	6	0	0
IIN	4	0	0	125	76	197	185	68	60	17	17	65	4	0	0
IMGN	4	0	0	0	0	101	74	270	253	36	27	49	4	0	0
IMGN	8	0	0	2	2	134	108	192	181	28	21	41	3	0	0
IMH	1	0	0	22	13	329	326	45	31	17	17	21	2	3	0
IMMU	5	0	0	0	0	325	324	28	22	3	3	7	0	0	0
ING	10	0	0	6	6	21	5	206	200	123	77	68	0	0	0
INOD	9	2	0	106	97	117	99	36	30	15	10	34	5	1	0
INS	10	5	5	112	79	152	123	35	31	11	9	59	9	0	0

Code	# of Epochs to Best	Significant Drop		Moderate Drop		Stable		Moderate Increase		Significant Increase		Incorrect – Magnitude of Error			
		Total	Corr.	Total	Corr.	Total	Corr.	Total	Corr.	Total	Corr.	1	2	3	4
INSG	7	79	71	99	98	22	14	13	9	0	0	19	2	0	0
IPAS	2	109	100	87	69	12	10	5	4	0	0	30	0	0	0
ISIG	10	0	0	62	40	177	158	60	56	16	11	37	13	0	0
JAKK	8	5	3	80	74	70	39	53	45	5	4	41	7	0	0
JCS	7	1	1	84	66	187	119	117	112	18	16	71	14	8	0
JOB	3	0	0	77	50	266	234	62	56	8	7	64	2	0	0
JPM	5	9	2	83	51	185	153	117	101	13	11	77	11	1	0
KGC	4	93	87	111	96	7	6	1	0	1	1	23	0	0	0
KGJI	3	0	0	0	0	279	270	28	21	8	7	14	3	0	0
KIQ	1	0	0	44	5	276	158	70	64	17	12	139	29	0	0
KMT	5	0	0	0	0	4	0	32	12	320	320	22	2	0	0
KVHI	4	254	253	20	19	2	2	0	0	0	0	2	0	0	0
LEE	7	0	0	74	71	289	261	40	29	10	10	41	1	0	0
LFVN	1	0	0	40	24	241	161	62	53	13	1	105	11	1	0
LPX	8	23	2	243	228	102	81	34	29	11	9	57	3	4	0
LSCC	10	7	7	103	75	162	129	73	62	11	10	61	12	0	0
LTBR	1	0	0	16	14	295	295	2	1	2	2	3	0	0	0
LYTS	8	46	40	168	147	51	48	8	4	3	3	34	0	0	0
MAGS	4	10	8	245	234	40	33	16	13	4	4	20	3	0	0
MAMS	5	0	0	6	2	73	50	180	163	17	16	27	14	4	0
MERC	2	15	15	252	236	122	82	14	11	10	9	54	5	1	0
MFIN	10	27	22	188	166	139	124	22	18	12	10	48	0	0	0
MICR	2	0	0	15	14	251	202	123	101	18	18	68	4	0	0
MIND	9	0	0	12	4	74	34	246	233	81	69	64	9	0	0
MNI	9	0	0	83	78	197	189	24	18	11	9	21	0	0	0
MNTA	7	0	0	64	44	106	100	34	21	9	7	39	2	0	0
MSTR	6	0	0	30	25	254	246	64	56	8	4	23	2	0	0
MXC	9	0	0	104	48	283	276	14	7	6	6	67	2	1	0
MXWL	6	0	0	20	3	76	39	228	214	64	58	58	10	6	0
NAII	7	0	0	29	26	136	98	185	161	38	32	63	6	2	0
NSYS	7	8	8	64	24	156	107	70	65	17	14	72	19	6	0
NTZ	8	0	0	17	6	98	75	128	121	33	30	30	9	5	0
NVLN	8	145	102	153	140	11	11	6	4	0	0	53	5	0	0
NYMX	10	0	0	64	31	255	248	87	64	7	7	58	5	0	0
NYNY	7	0	0	19	6	170	161	118	108	8	1	35	4	0	0
OI	10	260	257	15	14	0	0	1	1	0	0	4	0	0	0
OSUR	6	0	0	24	19	158	121	163	150	43	32	59	6	1	0

Code	# of Epochs to Best	Significant Drop		Moderate Drop		Stable		Moderate Increase		Significant Increase		Incorrect – Magnitude of Error			
		Total	Corr.	Total	Corr.	Total	Corr.	Total	Corr.	Total	Corr.	1	2	3	4
PAR	3	103	96	140	120	25	20	4	1	4	2	36	0	1	0
PDLI	7	0	0	43	33	246	203	61	58	6	4	51	7	0	0
PESI	5	0	0	5	5	30	27	131	128	47	37	16	0	0	0
PICO	3	0	0	0	0	11	1	94	48	308	306	51	7	0	0
PIR	10	0	0	19	16	241	238	11	10	5	5	7	0	0	0
PKD	9	96	58	225	193	72	61	9	4	5	5	72	14	0	0
PLX	4	0	0	0	0	264	262	7	7	5	5	2	0	0	0
PPIH	8	17	9	265	212	92	81	25	12	8	8	74	7	4	0
PRAN	3	1	1	63	39	101	93	40	29	8	6	43	2	0	0
PRCP	4	0	0	0	0	3	0	93	40	260	258	55	3	0	0
QCOM	7	0	0	26	18	250	230	118	105	19	12	48	0	0	0
QUMU	10	5	4	81	75	87	82	34	27	6	5	18	2	0	0
RADA	5	0	0	45	20	241	220	90	80	31	29	49	9	0	0
RAND	3	0	0	107	100	86	53	13	9	7	7	42	2	0	0
RAVE	9	6	6	177	160	168	129	52	33	10	9	73	3	0	0
RCMT	2	223	195	164	130	17	8	9	3	0	0	68	8	1	0
RELL	5	56	20	182	168	61	58	13	8	3	3	34	21	3	0
RLH	2	1	1	5	3	55	41	130	121	22	18	28	1	0	0
SCKT	10	0	0	83	58	146	137	39	33	8	7	41	0	0	0
SCX	10	15	7	123	68	179	135	80	79	16	15	75	28	6	0
SFE	4	0	0	5	0	82	74	230	204	39	31	41	6	0	0
SGMA	10	0	0	20	6	268	241	106	79	19	17	51	17	2	0
SGU	5	11	11	46	39	179	154	109	94	11	9	49	0	0	0
SIGA	10	0	0	39	27	258	210	97	84	19	19	66	4	3	0
SIGM	9	0	0	80	52	212	196	91	76	24	1	77	4	1	0
SMRT	6	0	0	3	3	167	165	30	24	13	13	8	0	0	0
SPB	7	1	0	11	2	61	41	235	217	48	43	37	9	6	1
SRDX	4	13	11	117	114	64	47	14	13	5	5	23	0	0	0
SRI	10	4	1	67	55	139	130	54	39	12	5	41	5	0	0
STAR	4	5	5	33	20	199	184	22	14	17	17	21	12	3	0
STRM	5	0	0	0	0	2	0	252	197	102	96	61	2	0	0
SYPR	10	0	0	10	4	201	137	139	133	38	24	87	3	0	0
TACT	1	0	0	78	27	197	153	119	98	19	16	93	22	4	0
TCI	8	3	3	80	54	191	165	90	80	24	22	51	13	0	0
THC	9	10	8	162	144	182	170	44	29	15	14	44	2	2	0
TIVO	4	28	23	132	125	41	33	11	10	1	0	21	1	0	0
TRK	10	0	0	0	0	10	5	117	101	229	212	37	1	0	0

Code	# of Epochs to Best	Significant Drop		Moderate Drop		Stable		Moderate Increase		Significant Increase		Incorrect – Magnitude of Error			
		Total	Corr.	Total	Corr.	Total	Corr.	Total	Corr.	Total	Corr.	1	2	3	4
TRT	8	0	0	70	15	161	118	124	120	33	23	75	34	3	0
TTMI	5	312	312	1	1	2	0	0	0	0	0	0	2	0	0
TWMC	6	65	43	248	236	59	51	24	18	11	4	51	4	0	0
UAMY	1	0	0	174	169	31	17	4	2	4	0	21	0	4	0
UCTT	7	0	0	9	6	87	59	99	98	18	12	35	3	0	0
UIS	7	1	1	148	133	148	140	83	76	8	8	24	6	0	0
UNM	5	0	0	14	10	25	11	217	156	100	99	69	7	4	0
USAK	7	79	45	250	217	50	38	20	16	8	8	82	1	0	0
USAS	6	18	18	162	160	27	18	3	3	3	2	12	0	0	0
USAU	3	70	64	166	156	24	18	13	10	3	1	27	0	0	0
USEG	3	0	0	114	97	190	169	90	70	19	16	60	1	0	0
USG	2	0	0	26	1	135	111	217	185	29	22	83	5	0	0
VGZ	10	10	8	154	153	34	22	12	11	3	2	16	1	0	0
VICR	10	126	112	209	192	54	27	16	8	2	1	58	8	1	0
VIRC	7	336	328	44	29	4	4	4	1	0	0	25	1	0	0
VOXX	9	0	0	62	24	224	220	99	75	28	22	63	9	0	0
VTNR	10	0	0	0	0	100	79	99	93	14	12	29	0	0	0
VVUS	10	0	0	153	79	162	155	81	56	11	9	96	7	5	0
WLK	3	197	195	15	12	1	1	0	0	0	0	5	0	0	0
WTT	5	0	0	127	71	209	185	46	44	6	4	64	20	0	0
WTW	9	0	0	24	21	197	192	43	39	12	9	15	0	0	0
WYY	9	165	163	34	28	6	6	6	4	2	2	10	0	0	0
X	10	31	17	205	194	105	71	37	35	10	9	59	3	0	0
XEC	8	14	12	126	115	100	75	34	31	2	2	38	3	0	0
ZIXI	10	0	0	4	0	157	108	197	183	30	12	77	8	0	0

A.4 Partitioned Convolutional Neural Network

Code	# of Epochs to Best	Significant Drop		Moderate Drop		Stable		Moderate Increase		Significant Increase		Incorrect – Magnitude of Error			
		Total	Corr.	Total	Corr.	Total	Corr.	Total	Corr.	Total	Corr.	1	2	3	4
ABB	10	0	0	21	20	26	11	225	216	43	16	50	2	0	0
AKS	3	29	1	213	197	45	12	24	11	4	1	63	20	10	0
BIOS	7	303	301	3	2	7	3	2	1	0	0	7	0	1	0
CIA	10	0	0	14	10	44	22	179	135	78	73	57	17	1	0
COT	8	0	0	39	36	264	255	5	0	7	6	16	2	0	0
CYBE	9	2	2	77	42	160	126	57	46	19	13	68	16	2	0
DYNT	9	0	0	105	75	207	192	61	41	15	14	57	6	3	0
EFOI	10	0	0	8	5	348	344	30	22	2	2	15	0	0	0
EMITF	10	30	29	189	160	99	63	35	29	3	1	66	8	0	0
FISV	8	68	14	235	233	69	20	13	7	3	0	108	5	0	1
GEOS	9	0	0	21	13	90	57	87	80	15	11	49	2	1	0
GFI	9	0	0	6	4	101	60	224	192	57	52	70	9	1	0
GILT	7	2	1	71	42	199	159	117	96	18	13	91	5	0	0
GPIC	10	0	0	127	109	158	92	20	17	10	3	85	8	1	0
GPX	10	0	0	0	0	12	8	77	43	267	257	44	4	0	0
GV	9	0	0	0	0	78	44	250	216	28	25	64	7	0	0
HDSN	10	0	0	45	29	199	168	90	70	22	21	43	22	3	0
HLIT	10	0	0	16	15	117	95	176	168	47	37	38	2	1	0
HMY	10	0	0	173	165	93	70	4	3	6	6	32	0	0	0
HOV	7	3	3	130	83	181	158	74	61	19	12	83	7	0	0
HRTX	9	47	18	202	190	52	27	10	6	4	4	67	3	0	0
ICAD	10	35	10	278	246	65	27	20	18	9	2	73	28	3	0
IDRA	7	3	0	341	331	50	30	15	10	4	0	41	1	0	0
IDSA	10	0	0	4	3	281	266	111	82	11	11	44	1	0	0
IESC	8	79	58	249	236	52	22	29	10	4	2	71	12	2	0
IGLD	9	0	0	12	9	157	119	114	99	32	22	66	0	0	0
IGLD	1	276	276	0	0	0	0	0	0	0	0	0	0	0	0
IIN	9	0	0	125	107	197	169	68	57	17	11	62	1	0	0
IMGN	10	0	0	2	2	134	106	192	164	28	25	56	3	0	0
IMGN	8	0	0	0	0	101	64	270	261	36	23	59	0	0	0
IMH	3	0	0	22	11	329	322	45	13	17	13	44	7	3	0
IMMU	9	0	0	0	0	325	314	28	20	3	3	17	2	0	0
ING	9	0	0	6	6	21	5	206	197	123	86	59	3	0	0
INOD	10	2	0	106	91	117	93	36	21	15	8	50	11	2	0
INS	9	5	5	112	59	152	108	35	28	11	7	88	19	1	0

Code	# of Epochs to Best	Significant Drop		Moderate Drop		Stable		Moderate Increase		Significant Increase		Incorrect – Magnitude of Error			
		Total	Corr.	Total	Corr.	Total	Corr.	Total	Corr.	Total	Corr.	1	2	3	4
INSG	8	79	70	99	93	22	4	13	7	0	0	30	8	1	0
IPAS	9	109	92	87	65	12	6	5	2	0	0	47	0	1	0
ISIG	7	0	0	62	53	177	163	60	37	16	15	38	9	0	0
JAKK	10	5	3	80	59	70	25	53	52	5	0	53	20	1	0
JCS	10	1	1	84	69	187	134	117	92	18	15	90	3	3	0
JOB	9	0	0	77	49	266	246	62	25	8	8	80	5	0	0
JPM	8	9	4	83	65	185	145	117	97	13	9	82	5	0	0
KGC	10	93	77	111	100	7	3	1	0	1	1	31	1	0	0
KGJI	9	0	0	0	0	279	274	28	18	8	4	17	2	0	0
KIQ	8	0	0	44	10	276	184	70	61	17	12	99	38	3	0
KMT	8	0	0	0	0	4	0	32	18	320	318	20	0	0	0
KVHI	6	254	245	20	13	2	2	0	0	0	0	15	1	0	0
LEE	6	0	0	74	60	289	279	40	32	10	8	33	1	0	0
LFVN	10	0	0	40	37	241	220	62	40	13	8	39	11	1	0
LPX	6	23	3	243	235	102	26	34	33	11	6	99	7	4	0
LSCC	9	7	0	103	60	162	135	73	61	11	8	78	12	2	0
LTBR	7	0	0	16	13	295	292	2	2	2	2	6	0	0	0
LYTS	9	46	19	168	146	51	31	8	3	3	3	61	11	0	2
MAGS	10	10	5	245	237	40	18	16	7	4	4	37	6	1	0
MAMS	8	0	0	6	1	73	29	180	160	17	12	58	14	2	0
MERC	8	15	15	252	233	122	83	14	11	10	8	60	3	0	0
MFIN	10	27	23	188	164	139	125	22	13	12	9	51	3	0	0
MICR	5	0	0	15	5	251	187	123	85	18	14	103	13	0	0
MIND	9	0	0	12	6	74	48	246	235	81	62	57	5	0	0
MNI	10	0	0	83	73	197	181	24	16	11	9	35	1	0	0
MNTA	10	0	0	64	49	106	98	34	24	9	8	33	1	0	0
MSTR	10	0	0	30	27	254	231	64	49	8	6	40	3	0	0
MXC	10	0	0	104	49	283	280	14	10	6	5	60	0	3	0
MXWL	8	0	0	20	7	76	44	228	212	64	43	61	21	0	0
NAII	8	0	0	29	23	136	108	185	113	38	37	84	20	3	0
NSYS	8	8	8	64	29	156	121	70	61	17	12	76	8	0	0
NTZ	9	0	0	17	6	98	61	128	101	33	30	54	17	7	0
NVLN	2	145	122	153	120	11	7	6	1	0	0	56	5	4	0
NYMX	10	0	0	64	47	255	234	87	65	7	7	54	6	0	0
NYNY	9	0	0	19	11	170	143	118	103	8	7	50	1	0	0
OI	1	260	252	15	0	0	0	1	0	0	0	23	0	1	0
OSUR	9	0	0	24	20	158	115	163	125	43	32	85	11	0	0

Code	# of Epochs to Best	Significant Drop		Moderate Drop		Stable		Moderate Increase		Significant Increase		Incorrect – Magnitude of Error			
		Total	Corr.	Total	Corr.	Total	Corr.	Total	Corr.	Total	Corr.	1	2	3	4
PAR	10	103	78	140	113	25	20	4	1	4	0	57	7	0	0
PDLI	5	0	0	43	22	246	211	61	46	6	3	61	12	1	0
PESI	9	0	0	5	0	30	22	131	121	47	37	24	5	4	0
PICO	7	0	0	0	0	11	4	94	79	308	300	30	0	0	0
PIR	9	0	0	19	15	241	233	11	6	5	4	17	1	0	0
PKD	4	96	47	225	189	72	45	9	1	5	2	109	11	3	0
PLX	1	0	0	0	0	264	252	7	1	5	2	9	12	0	0
PPIH	9	17	9	265	248	92	71	25	7	8	7	56	9	0	0
PRAN	6	1	1	63	29	101	91	40	30	8	8	46	8	0	0
PRCP	10	0	0	0	0	3	2	93	61	260	257	36	0	0	0
QCOM	7	0	0	26	13	250	230	118	97	19	17	54	1	1	0
QUMU	10	5	4	81	74	87	76	34	22	6	5	29	2	1	0
RADA	10	0	0	45	43	241	230	90	73	31	29	31	1	0	0
RAND	10	0	0	107	84	86	75	13	11	7	6	36	1	0	0
RAVE	9	6	4	177	162	168	120	52	47	10	5	69	5	1	0
RCMT	10	223	183	164	156	17	7	9	1	0	0	60	6	0	0
RELL	10	56	37	182	164	61	49	13	9	3	3	37	9	6	1
RLH	10	1	0	5	3	55	36	130	120	22	18	34	0	1	1
SCKT	8	0	0	83	63	146	119	39	29	8	4	52	8	1	0
SCX	8	15	14	123	80	179	137	80	67	16	13	88	14	0	0
SFE	8	0	0	5	0	82	66	230	208	39	35	38	6	3	0
SGMA	9	0	0	20	18	268	246	106	78	19	15	46	8	2	0
SGU	10	11	3	46	32	179	155	109	91	11	9	57	5	3	1
SIGA	8	0	0	39	22	258	239	97	67	19	19	60	5	1	0
SIGM	9	0	0	80	62	212	195	91	73	24	14	55	7	1	0
SMRT	5	0	0	3	0	167	165	30	17	13	10	16	4	1	0
SPB	8	1	0	11	4	61	33	235	216	48	31	59	12	1	0
SRDX	7	13	13	117	104	64	48	14	12	5	4	29	3	0	0
SRI	10	4	2	67	50	139	111	54	45	12	6	49	9	4	0
STAR	6	5	5	33	19	199	189	22	17	17	16	24	3	2	1
STRM	9	0	0	0	0	2	0	252	212	102	83	59	2	0	0
SYPR	7	0	0	10	4	201	184	139	107	38	34	56	3	0	0
TACT	10	0	0	78	46	197	178	119	94	19	15	70	9	1	0
TCI	8	3	3	80	58	191	158	90	74	24	17	74	4	0	0
THC	8	10	8	162	138	182	160	44	25	15	13	63	6	0	0
TIVO	10	28	24	132	125	41	27	11	11	1	0	25	0	1	0
TRK	9	0	0	0	0	10	10	117	93	229	212	41	0	0	0

Code	# of Epochs to Best	Significant Drop		Moderate Drop		Stable		Moderate Increase		Significant Increase		Incorrect – Magnitude of Error			
		Total	Corr.	Total	Corr.	Total	Corr.	Total	Corr.	Total	Corr.	1	2	3	4
TRT	10	0	0	70	64	161	141	124	99	33	27	53	4	0	0
TTMI	4	312	311	1	0	2	2	0	0	0	0	2	0	0	0
TWMC	8	65	53	248	218	59	50	24	19	11	6	55	6	0	0
UAMY	3	0	0	174	166	31	16	4	0	4	0	24	3	4	0
UCTT	9	0	0	9	3	87	58	99	86	18	11	49	3	3	0
UIS	9	1	0	148	112	148	136	83	67	8	7	62	4	0	0
UNM	8	0	0	14	9	25	16	217	184	100	91	53	3	0	0
USAK	8	79	58	250	233	50	32	20	17	8	6	57	4	0	0
USAS	8	18	18	162	142	27	19	3	2	3	1	27	2	1	1
USAU	9	70	63	166	141	24	15	13	5	3	2	45	5	0	0
USEG	10	0	0	114	97	190	149	90	84	19	12	62	9	0	0
USG	9	0	0	26	22	135	94	217	205	29	25	56	5	0	0
VGZ	10	10	7	154	146	34	23	12	8	3	2	20	6	1	0
VICR	7	126	111	209	174	54	45	16	12	2	2	60	3	0	0
VIRC	10	336	330	44	24	4	4	4	3	0	0	26	1	0	0
VOXX	9	0	0	62	46	224	207	99	81	28	22	51	4	2	0
VTNR	7	0	0	0	0	100	83	99	83	14	7	37	3	0	0
VVUS	10	0	0	153	101	162	146	81	63	11	11	74	12	0	0
WLK	10	197	195	15	6	1	1	0	0	0	0	11	0	0	0
WTT	6	0	0	127	68	209	182	46	28	6	4	102	2	2	0
WTW	10	0	0	24	22	197	182	43	38	12	10	24	0	0	0
WYY	9	165	159	34	25	6	3	6	2	2	2	21	1	0	0
X	9	31	24	205	174	105	73	37	24	10	10	71	12	0	0
XEC	10	14	14	126	105	100	88	34	25	2	2	41	1	0	0
ZIXI	9	0	0	4	3	157	118	197	168	30	25	72	2	0	0